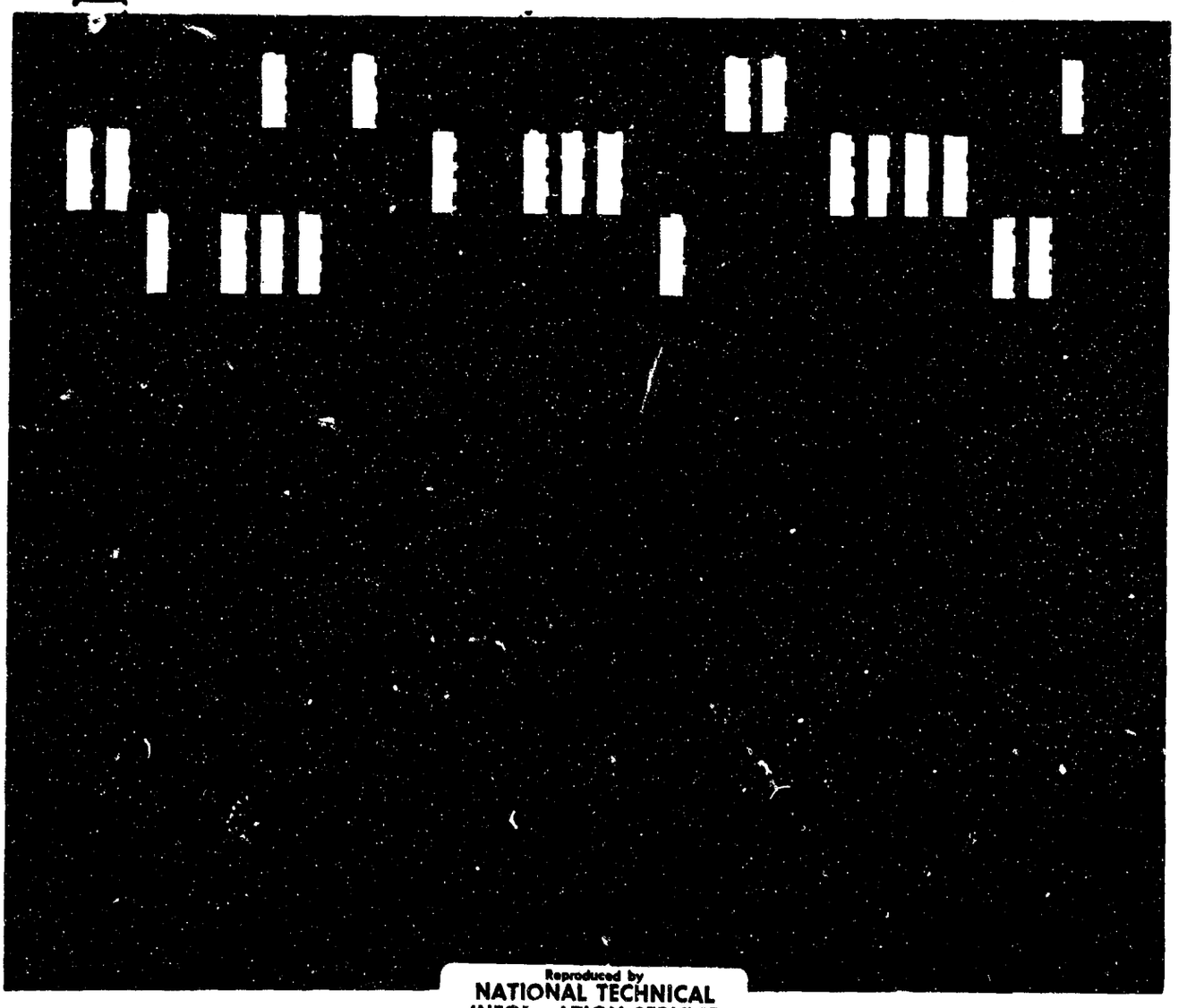
# UHTSS LIBRARY MANAGEMENT YESTERDAY, TODAY, AND TOMORROW

by

Alan C.H. Kam

University of Hawaii, Honolulu







THE ALOHA SYSTEM
UNIVERSITY OF HAWAII
HONOLULU, HAWAII 96822

Technical Report B70-3

November 1970

1. This [illegible] for public release and [illegible] is unlimited.

Prepared under Grant Number F44620-69-C-0030 from the Air Force Office of Scientific Research (OAR); a Project THEMIS award.

85

# UHTSS LIBRARY MANAGEMENT
# YESTERDAY, TODAY, AND TOMORROW *

BY

ALAN C.H. KAM
UNIVERSITY OF HAWAII, HONOLULU

## ABSTRACT

THIS REPORT IS A COLLECTION OF INTERNAL REPORTS DEALING WITH THE LIBRARY MANAGEMENT. THE PRELIMINARY DESIGN DETAILS THE ANTICIPATED SYSTEM. THE STRUCTURE OF DATA BASE REVEALS THE IMPLEMENTATION SCHEME ON AN IBM 2314 DISK FACILITY. VARIOUS ALGORITHMS ARE PRESENTED TO DOCUMENT THE ACTUAL SYSTEM CONDITIONED BY USING XPL AND OS MVT/HASP. FINALLY A DETAILED DESCRIPTION OF THE XPL PROGRAM ELABORATES UPON THE MODULAR APPROACH.

*

THIS WORK IS A JOINT PROJECT OF THE UNIVERSITY OF HAWAII COMPUTING CENTER AND THE ALOHA SYSTEM, A RESEARCH PROJECT RESEARCH (SRMA) UNDER CONTRACT NUMBER F44620-69-C-0030, OF THE UNIVERSITY OF HAWAII, SUPPORTED BY THE OFFICE OF AEROSPACE A PROJECT THEMIS AWARD.

## ABSTRACT

THE UHTSS USER'S LIBRARY IS A COLLECTION OF USER'S DATA SETS WHOSE FORMAT IS EITHER SOURCE (80 BYTE CARD IMAGE RECORDS) OR LOADABLE (3000 BYTE CORE IMAGE RECORDS). EACH USER'S DATA SET IS AN INDEXED LIST OF LOGICAL TRACKS (3520 BYTES = 1/2 PHYSICAL TRACK). THE LIBRARY PROVIDES SIMPLE EDITING FUNCTIONS FOR SOURCE DATA SETS, ROUTES SOURCE DATA SETS TO EXTERNAL DEVICES, AND MANAGES THE DATA SETS. THIS PROGRAM IS WRITTEN IN XPL USING THE DIRECT ACCESS FEATURES. THE LIST PROCESSING ASPECTS OF ISS-LIBRARY ARE USER WRITTEN SINCE THE XPL FACILITIES ARE INADEQUATE. IN PARTICULAR, ARRAY MANIPULATION AND STRING MOVE FUNCTIONS ARE CODED AS XPL PROCEDURES.

THE MANAGEMENT OF THE LIBRARY CONSISTS OF "CREATING" DATA SETS, "SCRATCHING" DATA SETS, "SELECTING" DATA SETS, AND "NAMING" DATA SETS.

ONCE A DATA SET HAS BEEN CREATED/SELECTED. ONE MAY ENTER SOURCE CARDS ONTO HIS DATA SET. TWO EDITING FUNCTIONS PROVIDE A SIMPLE METHOD TO ACCOMPLISH THE UPDATING. A SOURCE CARD IS A 72 BYTE IMAGE WITH A 6 BYTE KEY. THE KEY SEQUENCES THE POSITION OF THE CARD WITHIN THE DATA SET. "INSERT" ADDS A CARD TO OR REPLACES A CARD IN THE DATA SET, WHILE "DELETE" REMOVES A CARD FROM THE DATA SET. BOTH CREATION AND INSERTION UTILIZE THE TRACK ALLOCATOR, WHILE SCRATCHING AND DELETION USE THE TRACK FREER. THE TRACK MANAGEMENT ROUTINES CHECK POINT THE IN-CORE TABLE ONTO THE DISK FILE EACH TIME THE TABLE ENTRIES ARE ALTERED.

NOW IF THE USER HAS FINALLY DEBUGGED HIS PROGRAM AND WANTS TO REVIEW IT, TSS-LIBRARY PROVIDES THIS FUNCTION. IN GENERAL, SOURCE DATA SETS CAN BE ROUTED TO THE BATCH PROCESSOR AND TO 3 EXTERNAL DEVICES - PRINTER, PUNCH, AND HIS TERMINAL. USING THE ROUTING ROUTINES, THE USER CAN OBTAIN A PERMANENT HARD COPY OF HIS PROGRAMS. THE RESPECTIVE COMMANDS ARE 'HASP', 'PRINT', 'PUNCH', AND 'DISPLAY'.

FINALLY THE USER MAY USE INTERNAL COMMANDS WHICH MANIPULATE THE CONTENTS LOGICAL TRACKS. HE CAN SELECT DATA SETS USING "LINK", READ A LOGICAL TRACK RECORD BY USING "LOAD", AND WRITE A LOGICAL TRACK RECORD BY USING "DAOL".

# CONTENTS

INTRODUCTION
- SYSTEM OVERVIEW
- DATA BASE OVERVIEW
- BACKGROUND HISTORY

COMMAND STRUCTURE
- DATA SET MANAGEMENT COMMANDS
- DISPLAY COMMANDS
- EDITING COMMANDS
- INTERNAL COMMANDS
- DATA SET ASSIGNMENT

PRELIMINARY DESIGN
- STRUCTURE OF THE ENTIRE LIBRARY
- DIRECTORY SCHEME
- LOADLIB AND USERLIB
- SYSTEM UPDATE SCHEME
- MANAGEMENT TECHNIQUES
- APPENDIX

DATA BASE STRUCTURE
- STRUCTURE OF OS/MVT IMPLEMENTATION
- STRUCTURE OF UHTSS USERLIB
- STRUCTURE OF DIRECTORY TABLE OF CONTENTS
- STRUCTURE OF DIRECTORY PAGE
- STRUCTURE OF UHTSS DATA SET
- APPENDIX

LIBRARY MANAGEMENT ALGORITHMS
- TRACK MANAGEMENT
- DATA SET MANAGEMENT
- JOB MANAGEMENT

XPL IMPLEMENTATION
- SOURCE LAYOUTS
- SUBROUTINE COMPONENTS
- SUBROUTINE ANATOMY

FILE MAINTENANCE ROUTINES
- ZERODISK
- STATDISK
- ANLZDISK

# INTRODUCTION

## SYSTEM OVERVIEW

WITH THE INSTALLATION OF THE MVT OPERATING SYSTEM ON THE UH IBM 360/65, THE TIMESHARING SYSTEM DEVELOPED HERE FOR USE WITH THE MODEL 50 BECAME OBSOLETE. A NEW SYSTEM, UHTSS, IS NOW UNDER DEVELOPMENT AS A JOINT PROJECT OF THE COMPUTING CENTER AND THE ALOHA SYSTEM. UHTSS CURRENTLY UTILIZES IBM 2260'S, AND SHOULD BE EASILY ADAPTABLE TO ANY KIND OF TERMINAL WE ACQUIRE.

UHTSS HAS A CONSOLE USER'S DATA FILE STRUCTURED TO FACILITATE EDITING FROM THE CONSOLE. DATA SETS OF CARD IMAGES CAN BE CREATED, EXAMINED AND EDITED FROM THE CONSOLE AND PRINTED, PUNCHED, OR READ BY A CONSOLE OR BACKGROUND PROGRAM. CONSOLE USER'S DATA SETS CAN ONLY BE ACCESSED BY UHTSS UTILITY ROUTINES, BECAUSE THEY DO NOT USE A STANDARD IBM ACCESS METHOD. THE SYSTEM WILL ALSO BE ABLE TO USE STANDARD IBM OS DATA SETS, BUT THE FULL FACILITY FOR THIS WILL REQUIRE CONSIDERABLE TIME TO DEVELOP.

PROGRAMS CAN BE EXECUTED IN THREE MODES. FIRST, THE VERY HEAVILY USED ROUTINES FOR DISPLAYING AND EDITING CONSOLE USERS' DATA SETS WILL BE IN THE MEMORY ALL THE TIME. SECONDLY, A PART OF THE MEMORY WILL BE AVAILABLE TO SHARE AMONG CONSOLE USERS BY "SWAPPING" TO THE DISK UNIT. THIS WILL BE USED FOR VERY SHORT PROGRAMS AND INTERACTIVE PROGRAMS. THIRDLY, A PERSON WILL BE ABLE TO PREPARE A PROGRAM IN MUCH THE SAME MANNER AS FOR ORDINARY BATCH OPERATION AND SUBMIT IT TO THE BATCH OPERATION BY COMMANDS FROM THE CONSOLE.

OUR OLD TIME-SHARING SYSTEM USED STANDARD BATCH COMPILERS. IT IS NOT OUR INTENTION TO INCLUDE THIS FACILITY WITH THE NEW SYSTEM, BUT TO PROVIDE COMPILERS OR INTERPRETERS ADAPTED PRIMARILY TO TIME-SHARING SYSTEMS. THE BATCH COMPILERS ARE MOST GENERAL BUT ARE AWKWARD AND SLOW FOR USE WITH THE 2260. FUTHERMORE, THEIR USE WITH A TYPEWRITER CONSOLE WILL BE MUCH MORE DIFFICULT. CURRENTLY, A BASIC COMPILER AND A DESK CALCULATOR FACILITY ARE AVAILABLE.

## DATA BASE OVERVIEW

THE PRELIMINARY DESIGN OF THE LIBRARY MANAGEMENT WAS BASED ON A GRANDIOSE VIEW OF AN INFORMATION RETRIEVAL SYSTEM. IT WAS CONDITIONED BY IBM'S DATA MANAGEMENT SYSTEMS AND IN PARTICULAR PARTITIONED DATA SETS. AFTER INVESTIGATING THE PDS SCHEME, THE FOLLOWING CONCLUSION WAS DRAWN: IT WON'T WORK. THE MAIN REASON FOR THIS STATEMENT WAS THAT THE PDS SCHEME LACKED DYNAMIC GARBAGE COLLECTION (WHICH IBM CALLS DEGASSING). THUS THIS FIRST EFFORT IN SPECIFYING WHAT WAS DESIRED REFLECTED ONLY ASPIRATIONS ON THE GROSSEST SCALE. YET, ITS LOGIC HAS PREVAILED WITHIN THE UHFSS LIBRARY SCHEME.

THE COMMAND STRUCTURE REFLECTED A MODEST ASPECT OF THE PRELIMINARY DESIGN. THE COMMANDS PROVIDED SIMPLE FUNCTIONS, MAINLY THOSE REQUIRED TO MANIPULATE CARD IMAGED DATA SETS. THE LIBRARY SCHEME WAS STRUCTURED TO THIS END.

THE DATA BASE STRUCTURE WAS A STRAIGHTFORWARD SCHEME DESIGNED TO MINIMIZE DISK ACCESSES. THE RETRIEVAL SCHEME WAS PREDICATED UPON THE EXISTANCE OF SEARCH KEYS. IN ORDER TO ACCESS A SINGLE CARD THE FOLLOWING KEYS WERE AT ONE TIME USED: THE JOB/ACCOUNT NUMBER KEY, THE DATA SET NAME KEY, AND THE CARD KEY. THIS LIBRARY PLAN DIFFERED FROM MANY TIME-SHARING SYSTEMS BY NOT COPYING THE USER'S FILE ONTO A WORK FILE, EDITING THAT FILE AND LATER RECOPYING THE EDITED ONE ONTO THE ORIGINAL, BUT BY PERFORMING ALL EDITING IN-PLACE.

WITH THIS DATA BASE, SEVERAL ALGORITHMS WERE DELEVLOPED TO MANAGE THE FILE. HALF OF THEM WERE QUITE NATURAL AND WERE DESIGNED BEFORE THE IMPLEMENTATION. THE OTHER HALF WERE BORN OUT OF NECESSITY.

THE XPL IMPLEMENTATION USED A HORRENDOUS COLLECTION OF SUBROUTINES. THE BASIC CONCEPT PREVALENT THROUGHOUT THE PROGRAM WAS THAT OF BLOCKS. BLOCKS WERE CREATED, MOVED ABOUT AND DELETED. THE VISUAL APPEARANCE OF THE PROGRAM WAS DESIGNED TO PROVIDE A PICTURE OF THE INTERNAL RELATIONSHIPS. SINCE THE FLOW OF CONTROL IS LINEAR, SOME EFFORT WAS EXPENDED TO LINEARIZE THE ALGORITHMS. NOTE THAT BOTH ERROR AND NORMAL EXITS WERE LOCALIZED.

THE LAST SEGMENT OF THE DATA BASE OVERVIEW IS THE FILE MAINTENANCE ROUTINES. TWO OF THESE PROGRAMS USE THE DISK AS A DIRECT ACCESS FILE, NOT A UHTSS LIBRARY FILE. THE OTHER PROGRAM, ANLZDISK, IS STILL IN ITS INFANCY. MUCH WORK IS REQUIRED IN THE AREA OF DISK SPACE ACCOUNTING, DEAD STORAGE MANAGEMENT, AND DATA SET REORGANIZATION.

## BACKGROUND HISTORY

THE BASIC REASON FOR CHOOSING THE XPL SYSTEM WAS THAT IT WAS A SYTEM THAT WE COULD UNDERSTAND, ALTER AND RE-DESIGN. WE DID NOT WANT TO PATCH EXISTING COMPLIER SYSTEMS.

ALTHOUGH FORTRAN H WOULD HAVE GIVEN US THE MOST EFFICIENT CODE, ITS STRING PROCESSING CAPABILITIES WERE NON-EXISTANT. STRING MANIPULATION IS NOT A REQUISITE BUT ADDS TO THE STRUCTURAL RELATIONSHIPS OF THE ROUTINES. PL/1 WITH ITS LIST PROCESSING WOULD HAVE BEEN PERFECT, HOWEVER, THE OVERHEAD IN LINKAGE, ET CETERA, WOULD HAVE BEEN A DISADVANTAGE. WITH THE IMPROVEMENTS AND THE COMMITMENT MADE BY IBM, PL/1 HAS IMPROVED AND PERHAPS WOULD HAVE BEEN THE BEST CHOICE.

ANOTHER REASON FOR CHOOSING XPL WAS BASED ON THE OBJECTIVES OF THE PROJECT, NAMELY TO INVESTIGATE HIGHER-LEVEL LANGUAGES FOR TIME-SHARING. OBVIOUSLY ONE MUST REVAMP THE COMPILER IN ORDER TO PROVIDE THE PROPER STRUCTURING. MANUFACTURER'S COMPILERS WERE CLEARLY TOO DIFFICULT TO ALTER, MUCH LESS TO OBTAIN IN SOURCE FORM. OUR RECOURSE WAS TO USE SMALL, FAST BOOTSTRAPPING COMPILERS. XPL WAS THE ONLY ONE AVAILABLE AND IT HAD CONSIDERABLE DOCUMENTATION.

THE ONLY CHANGE TO THE COMPILER, TO DATE, WAS THE ADDITION OF A SIMPLE EDITOR FACILITY. MOST SYSTEMS UPDATE THE MASTER FILE, PRODUCE A NEW MASTER FILE, AND THEN COMPILE THE NEW MASTER FILE. THIS INVOLVES AN EXTRA PASS THROUGH THE SOURCE AND EXTRA DISK/TAPE STORAGE. IT WAS CLEAR THAT THE COMPILER COULD BE EASILY MODIFIED TO PERFORM THE MERGE AND COMPILE AT THE SAME TIME. IN THIS MANNER A NEW MASTER FILE NEED NOT BE CREATED UNTIL ALL THE CHANGES WERE DEBUGGED.

THE UPDATE PROCESS INVOLVES A SERIES OF $MERGE CARDS AND SETS OF UPDATE CARDS. A SET OF UPDATE CARDS IS THE GROUP OF SOURCE CARDS BETWEEN A PAIR OF $MERGE CARDS. THESE SOURCE CARDS CONTAIN THE NORMAL XPL SOURCE STATEMENTS WITHOUT ANY OTHER REFINEMENTS.

ALTHOUGH THE $MERGE CARDS ARE QUITE PRIMITIVE, THE BASIC FUNCTION HAS PROVEN INVALUABLE IN OUR DEBUGGING. FOR ONE THING, WE DO NOT NEED TO LUG BOXES OF CARDS TO BE MANGLED IN THE READER. SECONDLY, WE NEED LESS OS INTERACTION WITH RESPECT TO DD CARDS IF WE USE STANDARD UPDATE PROCEDURES. FINALLY, LESS SPACE ON THE DISK IS REQUIRED SINCE ONLY "GOOD" VERSIONS WILL BE THERE.

## DATA SET MANAGEMENT COMMANDS

THE FOLLOWING COMMANDS MANAGE THE USER'S DATA SETS. THEY MAKE AUXILIARY STORAGE AVAILABLE TO THE USER FOR DATA OR PROGRAM RETRIEVAL.

| COMMAND | PARAMETERS | REMARKS |
|---|---|---|
| SELECT | DATA SET NAME | THE SELECTED OLD DATA SET IS USED FOR ENSUING COMMANDS SUCH AS DISPLAY, EDITING COMMANDS, OR INPUT FOR A COMPILER OR OTHER PROGRAM. |
| CREATE | DATA SET NAME | CREATE A NEW DATA SET IN THE CONSOLE USERS DATA FILE, AND SELECT IT. THE DATA SET WILL BE AUTOMATICALLY SCRATCHED AFTER AN ELAPSED PERIOD OF 30 DAYS. |
| NAME | OLD NAME, NEW NAME | RENAME DATA SET. THIS COMMAND IS USEFUL WHEN A PROGRAMMER DEVELOPES A ROUTINE IN A DATA SET 'TEST'. WHEN HIS IS FINALLY FINISHED DEBUGGING TO PROGRAM, HE CAN CHANGE THE NAME TO 'EIGEN'. |
| SCRATCH | DATA SET NAME | SCRATCH THE DATA SET FROM THE LIBRARY. IF THE DATA SET HAS BEEN CREATED AND SCRATCHED WITHIN A WEEK THEN THE USER WILL NOT BE CHARGED FOR THE AUXILIARY STORAGE SPACE. |

EXAMPLES OF EACH COMMAND FOLLOWS: NOTE THAT THE COMMANDS MAY BE ABBREVIATED AND THAT ONLY THE FIRST 8 CHARACTERS OF THE DATA SET NAME ARE USED.

```
SELECT JOHN

CREATE PETE

NA IGLD UNEW;

SCR MANY
```

## DISPLAY COMMANDS

THE FOLLOWING COMMANDS RELATE TO DISPLAY FUNCTIONS. AT ALL TIMES THE OPERATION IS ON THE LAST SELECTED DATA SET, AND FURTHERMORE, THE SYSTEM KEEPS TRACK OF THE LAST RECORD WHICH WAS REFERENCED BY A DISPLAY OR EDITING COMMAND. THEN A RECORD CAN BE DESIGNATED IN ANY OF SEVERAL WAYS.

| | |
|---|---|
| DISPLAY | IF NO PARAMETER IS GIVEN, STEP=1 IS ASSUMED. |
| DISPLAY ALL | DISPLAY ALL THE DATA SETS WITH THE SAME JOB NUMBER. |
| DISPLAY DATA SETS | DISPLAY ALL DATA SETS WITH THE SAME JOB NUMBER AND USER'S NAME. |
| DISPLAY FIRST | DISPLAY THE FIRST RECORD IN THE DATA SET. |
| DISPLAY KEY=X | DISPLAY THE RECORD WHOSE KEY IS X (1 TO 6 CHARACTERS). NOTE THAT ' 1' IS NOT THE SAME AS '01'. IN FACT '000001' IS NOT THE SAME TOO. |
| DISPLAY LAST | DISPLAY THE LAST RECORD IN THE DATA SET. |
| DISPLAY NUMBER=N | DISPLAY THE NTH RECORD IN THE DATA SET. |
| DISPLAY STEP=N | DISPLAY THE NTH RECORD AFTER THE LAST RECORD REFERENCED (OR BEFORE IF N IS NEGATIVE). IF THE NTH RECORD IS NOT WITHIN THE SAVE DISK RECORD, THE NTH RECORD BECOMES THE LAST CARD OF THE DISK RECORD (OR THE FIRST ONE). |
| DISPLAY SURF | DISPLAY YOUR JOB/ACCOUNT NUMBER, NAME, IOCCUNTS, CPU SECONDS, DATA SET NAME, AND CURRENT KEY. |
| DISPLAY TEXT='XXX' | SEARCH, STARTING FROM THE LAST RECORD REFERENCED, FOR THE CHARACTER STRING XXX, AND DISPLAY THE RECORD WHICH CONTAINS IT. |

## DISPLAY COMMANDS

TO ANY DISPLAY COMMAND, ANOTHER PARAMETER MAY BE ADDED WHICH IS A NUMBER WHICH DESIGNATES HOW MANY RECORDS SHOULD BE DISPLAYED (IF MORE THAN ONE). IF YOU DO NOT KNOW HOW MANY RECORDS ARE ON THE SELECTED DATA SET 'LAST' IS THE SAME AS 99999. FOR EXAMPLE, TO DISPLAY RECORDS 10, 11, 12, AND 13 OF THE SELECTED DATA SET USE THE FOLLOWING COMMAND:

```
DISPLAY NUMBER = 10, 4
```

TO SUBMIT A JOB FOR BATCH PROCESSING ONE ISSUES THE "HASP" COMMAND. THE PARAMETERS FOR "HASP" ARE THE SAME AS THOSE FOR "DISPLAY". THE ENTIRE OS JCL MUST BE PASSED WITH THIS COMMAND. FOR EXAMPLE:

```
HASP FIRST LAST
```

"PRINT" AND "PUNCH" MAY REPLACE THE "DISPLAY" VERB. "PRINT" ROUTES THE RECORD TO BE PRINTED WHILE "PUNCH" ROUTES THE RECORD TO BE PUNCHED. THE SYSTEM WILL INSERT THE APPROPRIATE JCL CARDS TO ACHIEVE THIS FUNCTION AND SUBMIT THE JOBS TO THE HASP SYSTEM. THUS THE FUNCTIONS (PRINT AND PUNCH) ARE LIMITED TO 2000 LINES AND 1000 CARDS. FOR EXAMPLE:

```
PRINT TEXT = 'AN APOSTROPHE '' IS OK.'''
```

NOTES: TO TERMINATE A DISPLAY IN EXECUTION, CAUSE AN INTERRUPT. IF THIS INTERRUPT IS DETECTED BY THE SYSTEM, THE DISPLAY WILL BE TERMINATED. LIKE THE OVERALL COMMAND STRUCTURE ONLY THE FIRST LETTER OF KEY WORDS ARE EXAMINED. S=1 AND S ARE DIFFERENT BECAUSE THE FORMER IS A KEYWORD (SIGNALLED BY THE EQUAL SIGN) AND THE LATTER IS A POSITIONAL PARAMETER. FINALLY, TO TERMINATE THE COMMAND SCANNER, ENTER A SEMI-COLON (;). THIS IS PARTICULARLY USEFUL FOR 2260'S WITH EXTRANEOUS DATA ON THE SCREEN.

## EDITING COMMANDS

FOR THE EDITING COMMANDS, EACH RECORD IN THE DATA SET MUST CONSIST OF A SIX CHARACTER KEY FOLLOWED BY 72 CHARACTERS OF INFORMATION SUCH AS FORTRAN STATEMENTS OR DATA. UTILITY PROGRAMS WILL BE AVAILABLE TO READ A CARD DECK INTO A DATA SET OR VICE VERSA. IN THAT CASE, IN READING, THE FIRST 72 COLUMNS WILL BE USED AS DATA, AND THE LAST 6 COLUMNS AS KEY, AND ON PUNCHING THE SAME FORMAT WILL BE USED.

| | | |
|---|---|---|
| DELETE | RECORD KEY | DELETE THE DESIGNATED RECORD. |
| DELETE | FIRST KEY, LAST KEY | DELETE RECORDS FROM THE FIRST KEY TO THE LAST KEY, INCLUSIVE. |
| DELETE | | DELETE THE LAST RECORD ACCESSED, E.G. THE LAST RECORD DISPLAYED. |
| IN | SIX CHARACTER KEY FOLLOWED BY 72 CHARACTER RECORD | ENTER THIS RECORD IN THE DATA SET:<br>A) IF THERE ARE NO OTHER RECORDS WITH THIS KEY, AS A NEW RECORD. OR,<br>B) IF ONE EXISTS WITH THIS KEY, AS A REPLACEMENT.<br><br>IN ORDER TO LIMIT THE ENTRY TO NO MORE THAN 80 CHARACTERS, THIS COMMAND HAS A DIFFERENT FORMAT FROM OTHERS:<br>1) THE TWO LETTER COMMAND "IN"<br>2) A KEY<br>3) 72 CHARACTERS OF TEXT<br><br>NOTE: THE KEY IS A STRING OF ONE TO SIX CHARACTERS. THE KEY WILL BE RIGHT-JUSTIFIED IF THE KEY IS LESS THAN SIX CHARACTERS (LEADING BLANKS ARE INSERTED INTO THE KEY TO MAKE IT SIX CHARACTERS). THE NEXT CHARACTER AFTER THE FIRST BLANK STARTS THE TEXT. IF THE KEY IS OF 6 CHARACTERS THEN THERE MUST BE NO INTERVENING BLANKS BETWEEN "IN" AND THE KEY, AND BETWEEN THE KEY AND THE TEXT. |

EDITING COMMANDS

THE FOLLOWING COMMAND ALLOWS THE USER TO ENTER INSERT MODE. INSERT MODE TAKES ANY INPUT WITHOUT SCANNING FOR UHTSS COMMANDS AND INSERTS THESE RECORDS ACCORDING TO THE KEY GENERATION FUNCTION. THE NORMAL MODE OF TERMINATION IS AN ENCOUNTER WITH THE SENTINEL.

| | | |
|---|---|---|
| INSERT | FIRST KEY,<br>INCREMENT,<br>SENTINEL | ENTER THE FOLLOWING RECORDS INTO THE DATA SET AUTOMATICALLY GENERATING THE KEYS. THIS PROCESS WILL STOP WHEN THE SENTINEL IS ENCOUNTERED.<br><br>THE "FIRST KEY" IS A NUMERIC STRING (ONLY 0 TO 9 ARE ALLOWED).<br><br>THE NEXT RECORD WILL HAVE A KEY WHICH IS THE PREVIOUS KEY PLUS THE "INCREMENT".<br><br>THE "SENTINEL" IS A STRING OF UP TO 8 CHARACTERS. THE DEFAULT STRING IS 'STOP ', THE FUNCTION OF THE SENTINEL IS TO TERMINATE THIS MODE OF INSERTION. THE FIRST 8 CHARACTERS OF THE RECORD ARE EXAMINED BEFORE IT IS INSERTED. IF THE SENTINEL STRING IS ENCOUNTERED, THE PROCESS STOPS. |
| INSERT | BASIC,<br>SENTINEL | THE STRING 'BASIC' MUST APPEAR TO ENTER THE INPUT BASIC MODE. IN THIS MODE THE BASIC STATEMENT LABEL BECOMES THE KEY OF THE CARD. THE SENTINEL STRING IS AS THE PREVIOUS INSERT COMMAND.<br><br>NOTE THE BASIC LABEL MUST OCCUR WITHIN THE FIRST 8 COLUMNS. THE FIRST BLANK TERMINATES THE SCAN OF THE LABEL. THE NEXT CHARACTER STARTS THE TEXT STATEMENT. IF THERE IS NO KEY WITHIN THE FIRST 8 COLUMNS, THE INSERT MODE IS TERMINATED. |

INTERNAL COMMANDS

THE FOLLOWING COMMANDS ENABLE SPECIAL SERVICES FOR PROGRAMS EXECUTING IN THE USER AREA.

| | | |
|---|---|---|
| LINK | DATA SET | SELECTS THE DATA SET AND TRANSFERS CONTROL INFORMATION TO THE USER'S BUFFER. |
| LOAD | INDEX | READS IN THE LOGICAL TRACK OF THE "LINKED" DATA SET CORRESPONDING TO THE INDEX VALUE. THE AREA MUST HAVE ITS ADDRESS IN AREALOC AND BE 3536 BYTES. |
| DAOL | KEY | WRITES A LOGICAL TRACK WITH THE CORRESPONDING KEY. THE AREA MUST HAVE ITS ADDRESS IN AREALOC AND BE 3536 BYTES. |

THE FOLLOWING COMMANDS ARE A RESULT OF IMPROVEMENTS TO THE DESIGN EXTENDING THE FLEXIBILITY OF THE SYSTEM.

| | |
|---|---|
| ALLOCATE | ALLOCATES A DATA SET NODE TO A TERMINAL, PROVIDING THE USER ACCESS TO MORE THAN ONE DATA SET. (NOT ISSUED AS A COMMAND). |
| FREE | FREE ALL DATA SET NODES EXCEPT THE LAST ONE. (SINCE IT IS PERMANENTLY ASSIGNED TO THE TERMINAL). |
| REMOVE | REMOVE A TERMINAL REQUEST FROM MY QUEUE PROPERLY. (RESULT OF 2260 IMPROVEMENTS). |

## DATA SET ASSIGNMENT

DURING EXECUTION A PROGRAM MAY REFERENCE UP TO SEVEN DIFFERENT DATA SETS. SEQUENTIAL, 80 BYTE RECORD I/O IS ACCOMPLISHED BY USING INPUT(1), OUTPUT(1), INPUT(2), OUTPUT(2). DIRECT ACCESS, 3000 BYTE RECORD I/O IS ACCOMPLISHED BY USING FILE(1), FILE(2) OR FILE(3). DEFAULT DATA SET ASSIGNMENTS ARE PROVIDED TO SATISFY NORMAL EXECUTION CONDITIONS. THEY ARE AS FOLLOWS:

| FILE | DEFAULT |
|---|---|
| OUTPUT(1) | TERMINAL |
| INPUT(2) | MOST RECENTLY SELECTED DATA SET |

BASIC USES INPUT(2) AS THE SOURCE OF THE PROGRAM TO BE COMPILED. CONSEQUENTLY, THE MOST RECENTLY SELECTED DATA SET PRIOR TO EXECUTING BASIC WILL BE AUTOMATICALLY SELECTED FOR COMPILATION. THE FIRST TIME AN UNSPECIFIED FILE IS REFERENCED FOR AN I/O OPERATION, THE USER IS REQUESTED TO DEFINE THE DATA SET BY THE MESSAGE

SPECIFY DATA SET FOR XXX.

WHERE XXX IS THE REFERENCED FILE. TO SPECIFY THE DATA SET, TYPE IN THE DATA SET NAME AND PRESS "ENTER". THE DATA SET SPECIFIED WILL BE SELECTED AND USED FOR ALL SUBSEQUENT OPERATIONS ON THAT FILE. TO SPECIFY THE TERMINAL AS THE DATA SET FOR ANY FILE, TYPE IN "//" AND PRESS "ENTER". AT THE TERMINATION OF EXECUTION, ALL REFERENCED DATA SET WILL BE DESELECTED AND KEPT.

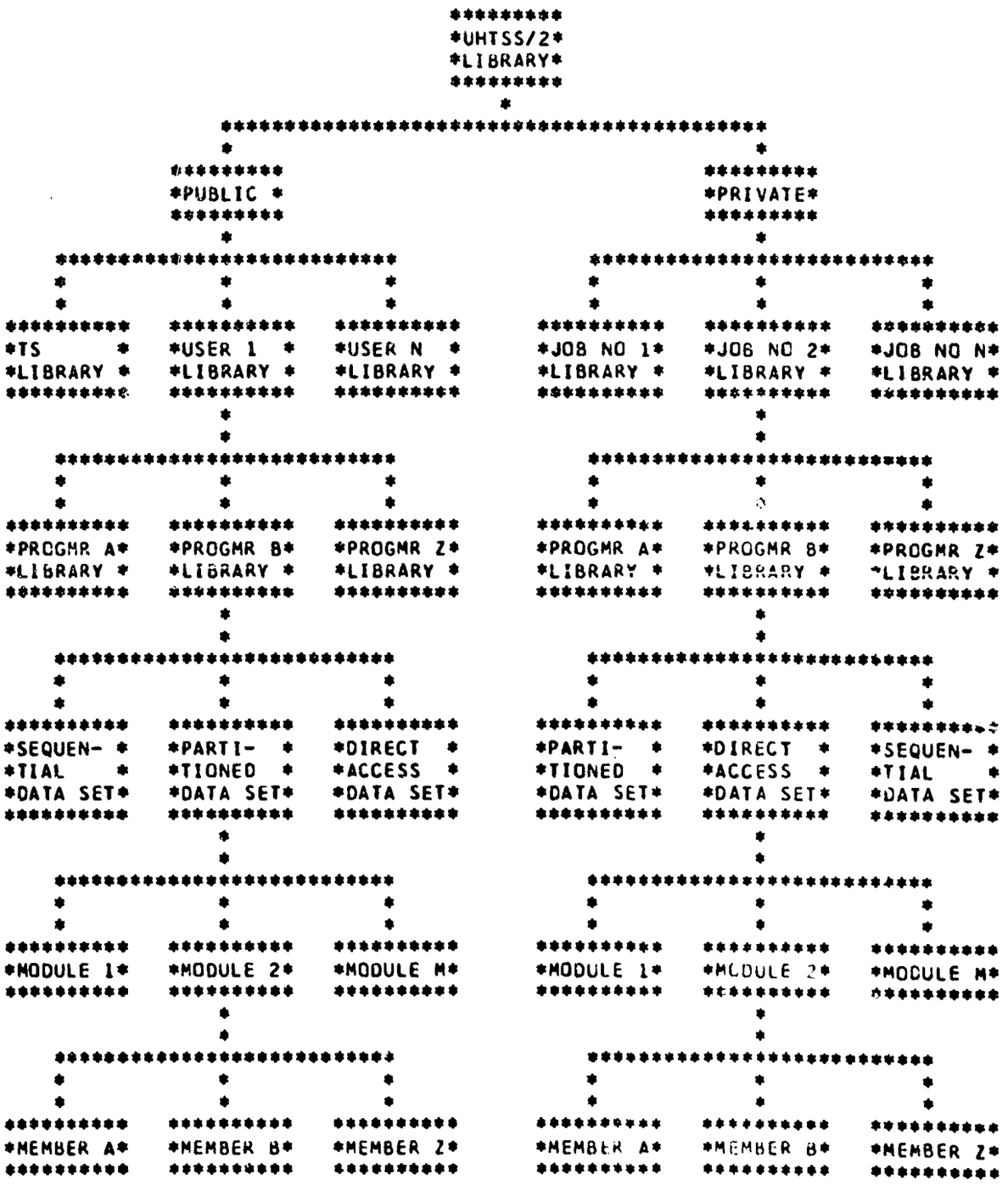
TO PREVENT DEFAULT ASSIGNMENTS, TYPE IN "U" AFTER THE PROGRAM NAME TO BE EXECUTED; SEPARATED FROM THE PROGRAM NAME BY BLANKS OR A COMMA.

## SIRUCIURE

THE UHTSS/2 LIBRARY STRUCTURE ALLOWS VARIOUS TYPES OF DATA SETS INCLUDING PARTITIONED DATA SETS, SEQUENTIAL DATA SETS AND DIRECT ACCESS DATA SETS. EACH OF THESE DATA SETS MAY CONTAIN SEVERAL USER LIBRARIES OR THERE MAY BE A ONE TO ONE CORRESPONDENCE. THE ALLOCATION OF THESE DATA SETS IS A FUNCTION OF THE UHTSS/2 MANAGEMENT.

PUBLIC LIBRARIES ARE DATA SETS THAT ARE MAINTAINED BY THE UHTSS/2 SYSTEM AND ARE AVAILABLE TO A LARGE NUMBER OF USERS WITH RELATIVE SECURITY. PRIVATE LIBRARIES ARE DATA SETS MAINTAINED BY THE USERS THEMSELVES AND ARE ALLOCATED IN COOPERATION WITH THE UHTSS/2 GROUP. THE SECURITY OF THESE DATA SETS IS USER DEFINED, (EITHER USING THE SYSTEM SCHEME OR PROVIDING THEIR OWN PASSWORD PROCESSOR). GARBAGE COLLECTION ON PRIVATE LIBRARIES IS A RESPONSIBILITY OF THE USER, WHILE GARBAGE COLLECTION ON PUBLIC LIBRARIES IS A RESPONSIBILITY OF UHTSS/2. USERS WITH PRIVATE LIBRARIES MAY USE OPERATING SYSTEMS FUNCTIONS TO MAINTAIN AND UPDATE THEIR FILES IF THESE SYSTEMS FUNCTIONS ARE CAPABLE OF SUCH MAINTENANCE. UHTSS/2 IS NOT REQUIRED TO MAINTAIN COMPATIBILITY WITH ANY PRIVATE FILES, HOWEVER THOSE FILES THAT ARE CURRENTLY UHTSS/2 COMPATIBLE WILL BE COMPATIBLE WITH FUTURE VERSIONS OF UHTSS/2.

EACH USER LIBRARY MAY HAVE ASSOCIATED PROGRAMMER'S LIBRARIES. THE PROGRAMMER'S LIBRARIES MAY CONSIST OF MANY DATA SETS EACH OF WHICH MAY CONTAIN MORE THAN ONE MODULE. THE USUAL CASE IS ONE MODULE PER DATA SET PER PROGRAMMER LIBRARY. EACH MODULE MAY HAVE MORE THAN ONE MEMBER, THIS IS THE CASE FOR A MODULE THAT IS A PDS.


UHTSS/2
LIBRARY
PUBLIC
PRIVATE
TS LIBRARY
USER 1 LIBRARY
USER N LIBRARY
JOB NO 1 LIBRARY
JOB NO 2 LIBRARY
JOB NO N LIBRARY
PROGMR A LIBRARY
PROGMR B LIBRARY
PROGMR Z LIBRARY
SEQUEN- TIAL DATA SET
PARTI- TIONED DATA SET
DIRECT ACCESS DATA SET
MODULE 1
MODULE 2
MODULE M
MEMBER A
MEMBER B
MEMBER Z

## UHTSS/2 DIRECTORY

THE DIRECTORY IS A TABLE OF CONTENTS OF ALL MODULES WITHIN THE PUBLIC LIBRARY OR INDICATORS TO THESE MODULES PROVIDING SUCH INFORMATION. THIS FACILITY IS REQUIRED FOR USERS SINCE THEY DO NOT HAVE EASY ACCESS TO THE LIBRARIES IN GENERAL. USERS MUST BE ABLE TO HAVE ALL DATA SETS ASSOCIATED TO THEIR ACCOUNTS LISTED YET LOCKING OUT THOSE WHO DO NOT HAVE THE SAME ACCOUNT. THE USER IS REQUIRED TO KEEP TRACK OF HIS PRIVATE LIBRARY. UHTSS/2 WILL NOT SEARCH PRIVATE LIBRARIES IN ORDER TO MAINTAIN THEIR INTEGRITY.

## UHTSS/2 SYSTEMS LIBRARIES

THE TWO PUBLIC UHTSS/2 SYSTEMS LIBRARIES ARE UHTSS/2 LOADLIB, A PDS WHICH CONTAINS ALL UHTSS/2 MONITOR ROUTINES AND USER'S PUBLIC LOAD MODULES, AND UHTSS/2 USERLIB, A DIRECT ACCESS DATA SET WHICH CONTAINS USER PROGRAMS IN EITHER SOURCE OR OBJECT FORM AND A LIMITED NUMBER OF USER DATA FILES (RESTRICTED TO SEQUENTIAL CARD IMAGE DATA THAT IS NORMALLY PASSED THROUGH THE SYSIN DATA SET).

## UHTSS/2 LOADLIB

THE UHTSS/2 LOADLIB IS A PARTITIONED DATA SET, PDS, WITH A PRIMARY ALLOCATION OF 50 CYLINDERS. ANY USER MAY ACCESS THIS LIBRARY SINCE THE LINKEDIT SYSLIB DD CARD WILL POINT TO THIS DATA SET, I.E. THE LIBRARY WILL BE CONCATENATED TO LOAD.SYSLIB. THIS LIBRARY WILL BE READ-ONLY EXCEPT UNDER SPECIAL CONDITONS. THE SPECIAL CONDITIONS WILL BE GOVERNED BY THE UHTSS/2 MANAGEMENT TO EFFECTIVELY SUPERVISE THE LOAD MODULES AND SCRUPULOUSLY RECOVER THE UPDATE PGM OPERATES BY EXAMINING THE MODULE SPECIFIED AND THE OPTIONS.

## UHTSS/2 USERLIB

THE UHTSS/2 USERLIB IS A DIRECT ACCESS OS DATA SET OF 50 CYLINDERS. ONLY USERS WHO HAVE THE PROPER PASSWORD MAY ACCESS THIS LIBRARY. LOCK-OUT FEATURES LIKE READ-ONLY TO ANY JOB NUMBER OR TO NO OTHER JOB NUMBER OR TO NO OTHER PROGRAMMER NAME WILL BE IMPLEMENTED. ONE PASSWORD SCHEME IS JOB NUMBER CONCATENATED WITH PROGRAMMER NAME CONCATENATED WITH THE MODULE NAME. ADDITIONAL FLAGS WILL BE SET WITHIN THE MODULE TO INDICATE ITS OTHER LOCK-OUT FEATURES. THE ACTUAL STRUCTURE OF THE USERLIB FOLLOWS:

A MODULE IS A LOGICAL RECORD.

A LOGICAL RECORD IS ONE OR MORE PHYSICAL TRACKS OF AN IBM 2314 DISK UNIT, WHICH HAS A DATA TRANSFER RATE OF 111 MS PER TRACK, ASSUMING THE AVERAGE SEEK AND ROTATION TIME. A PHYSICAL TRACK CONTAINS 7200 BYTES. IF THE LOGICAL RECORD CONTAINS MORE THAN A TRACK, A CHAIN POINTER WILL BE SET.

A LOGICAL RECORD CONSISTS OF ONE OR MORE LOGICAL MEMBERS. A LOGICAL MEMBER HAS THREE FLAGS, THE LANGUAGE FLAG, THE FORMAT FLAG, AND THE TYPE FLAG. THE LANGUAGE FLAG INDICATES WHETHER THE MEMBER IS A FORTRAN PROGRAM OR SOME OTHER HIGH-LEVEL LANGUAGE. THE TYPE FLAG INDICATES WHETHER THE MEMBER IS SOURCE DECK, AN OBJECT DECK, OR A DATA DECK. THE FORMAT FLAG INDICATES FORMAT OF THE MEMBER, WHICH IS EITHER CARD IMAGE, PRESSDECK, OR FREEFORM.

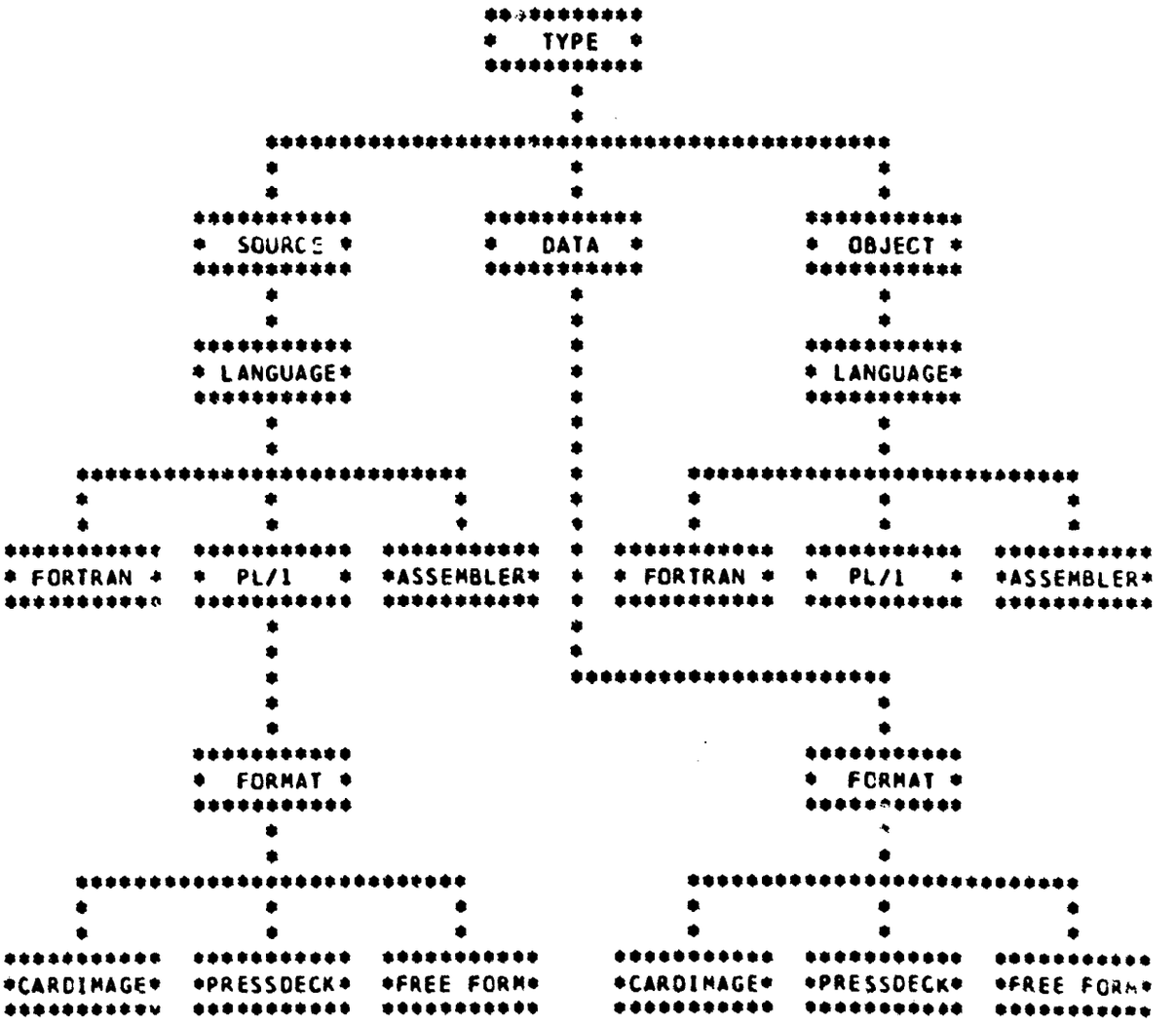
CARD IMAGE IS EXACTLY AS A CARD WOULD APPEAR, 80 BYTES OF EBCDIC INFORMATION.

PRESSDECK IS A COMPRESSION OF AN ACTUAL CARD WITH STRINGS OF THE SAME CHARACTER COMPRESSED INTO 3 BYTES OF CODED INFORMATION.

FREEFORM ALLOWS HIGH-LEVEL LANGUAGES TO BE EXPRESSED IN A SIMPLE SYNTAX. THIS FORMAT ALLOWS THE USER SOME INDEPENDENCE OVER HIS TERMINAL DEVICE AND FREES HIM FROM FIXED FORM WHERE COLUMN 7 IS NOT READILY VISIBLE. WHEN A PROGRAM IN FREEFORM IS PASSED TO A COMPILER, UHTSS WILL REFORMAT THE DATA INTO ACCEPTABLE FORMAT, NAMELY PROPER COLUMNS AND SO FORTH. THE FREEFORM SYNTAX IS SPECIFIED IN BACKUS NAUR FORM. NOTE THAT STRING IS ASSUMED TO BE THE CORRECT SEQUENCE OF CHARACTERS ALLOWED BY THE HIGH-LEVEL LANGUAGE.

```
<LABEL> := <NULL> | <STRING> | <LABEL>:
<COMMENT> := * <STRING>
<STATEMENT> := <STRING> | <COMMENT>
<SENTENCE> := <LABEL> <STATEMENT>;
<PROGRAM> := <SENTENCE> | <PROGRAM> <SENTENCE>
```

## SUMMARY OF FLAGS



## UHISS/2 UPDATE

THE FUNCTIONS OF THE UPDATE PROGRAM ARE INCLUDED IN THE FOLLOWING LIST:

1. INTERROGATE THE INPUT DATA SET WHICH MAY BE A PDS OR AN ISAM FILE.
2. POSITION DATA SET AT A DESIRED LINE NUMBER OR CHARACTER STRING.
3. UTILIZE THE WORK AREA DATA SET TO OPTIMIZE THE OTHER FUNCTIONS.
4. PROVIDE RECOVERY MECHANISMS FOR RETRIEVAL OF FILES AFTER SYSABEND.
5. ADD OR DELETE CHARACTER STRINGS WITHIN THE GIVEN TEXT.
6. OUTPUT TO THE USER RESULTS OF INTERROGATION (HARDCOPY/DISPLAY).
7. EDIT TEXTS BY REPLACING ONE STRING FOR ANOTHER STRING THROUGHOUT THE TEXT.

ACCIDENTAL DELETIONS OF MODULES:

IF A PROGRAM ID IS SPECIFIED, UPDATE PGM ATTACHES IT AND WAITS FOR ITS COMPLETION. NOTE THAT THE PROGRAM MUST RESIDE IN THE LIBRARY SPECIFIED. IF NO LIBRARY IS SPECIFIED, UHISS/2 LOADLIB IS USED. IF NO PROGRAM ID IS SPECIFIED, THE UPDATE PGM ATTACHES A MODULE WHICH WILL OPTIMIZE THE FUNCTIONS BY USING THE BEST ACCESS METHOD FOR THE GIVEN INPUT DATA SET AND WORK AREA.

## MANAGEMENT TECHNIQUES

SELECTIVE LOADING ALLOWS THE SOPHISTICATED USER TO SAVE TIME BY ELIMINATING RECOMPILATIONS AND SOFTWARE GENERATION.

SUPPOSE USER X HAS HIS FORTRAN PROGRAM IN SOURCE FORM. THE PROGRAM CONSISTS OF MAIN ROUTINE AND 3 SUBROUTINES, SUB1, SUB2, AND SUB3. NOTE THAT MAIN CALLS SUB1 WHICH CALLS SUB2 WHICH CALLS SUB3 WHICH CALLS MULREG. HE HAS COMPILED ALL OF THE MEMBERS OF THE MODULE AND SAVED THEM IN ANOTHER MODULE, SAY OBJTMOD. BUT MULREG HAS BEEN WRITTEN AND DEBUGGED BY USER Y AND HAS INSERTED INTO THE PUBLIC LIBRARY. ASSUME THAT SUB2 HAS A BUG IN IT . THE PROBLEM IS HOW DOES USER X CORRECT SUB2 AND EXECUTE THE PROGRAM AGAIN.

THE SOLUTION IS GIVEN IN A PROTO-TYPE OF THE COMMAND LANGUAGE. NOTE THAT MULREG IS AUTOMATICALLY LINKED WITH USER X'S PROGRAM.

```
UPDATE    0400.X.FORTMOD.SUB2
.
.         UPDATE SEQUENCE AS REQUIRED.
.
FORT      0400.X.FORTMOD.SUB2
SAVE      0400.X.OBJTMOD.SUB2
INCLUDE   0400.X.OBJTMOD
ENTRY     MAIN
XEQ
```

AN EXTENSION OF THE XEQ COMMAND WILL ALLOW THE USER TO COMPILE, LOAD, AND GO WITH A SINGLE INSTRUCTION. SUPPOSE USER X HAS A MODULE WITH 3 MEMBERS. THE FIRST MEMBER IS THE FORTRAN MAIN PROGRAM IN SOURCE FORM. THE SECOND MEMBER IS A FORTRAN SUBROUTINE IN OBJECT FORM. AND THE THIRD MEMBER IS DATA IN CARD IMAGE FORMAT.

THE INSTRUCTION XEQ MYMOD IS EQUIVALENT TO THE FOLLOWING INSTRUCTIONS. LET THE MODULE NAME BE "MYMOD" AND THE MEMBERS BE "MAIN," "SUB," AND "SYSIN." NOTE THAT "SYSIN" WOULD BE AUTOMATICALLY INSERTED INTO THE INPUT STREAM.

```
FORT      0400.X.MYMOD.MAIN
INCLUDE   0400.X.MYMOD.SUB
ENTRY     MAIN
XEQ
```

SWITCH NAME IS A TECHNIQUE THAT ALLOWS THE RECOVERY OF MODULES THAT HAVE BEEN DELETED. WHEN AN OLD MEMBER OF A MODULE IS UPDATED, ITS NAME IS CHANGED AND IT IS NOT SCRATCHED. THE NEW MEMBER IS INSERTED WITH THE PROPER NAME. IN THIS MANNER, THE GARBAGE COLLECTOR MAY RETRIEVE MODULES THAT HAVE BEEN ACCIDENTLY LOST. A PROBLEM IS THAT AN UNUSUAL AMOUNT OF DEAD SPACE IS CREATED, SO SHOULD THE UHTSS MANAGEMENT ALWAYS SCRATCH OLD MODULES AT THE END OF A STEP, OR SHOULD IT ALWAYS SAVE MODULES UNTIL THE END OF THE JOB.

PASSWORD SECURITY ALLOWS THE USER LOCK-OUT OF USERS WHO DO NOT SHARE THE SAME JOB NUMBER OR OTHER ESOTERIC CONVENTIONS. THE PROPOSED TECHNIQUE IS A METHOD OF PASSWORD SECURITY, WHERE THE PASSWORD IS THE CONCATENATED STRING OF CHARACTERS INCLUDING THE JOB NUMBER, THE PROGRAMMER'S NAME, AND THE MODULE NAME. THE NEXT LEVEL OF PROTECTION IS TO INCLUDE A MODULE PASSWORD AND FINALLY THE MEMBER PASSWORD. BUT THERE IS STILL A NEED FOR THE ABILITY TO ALLOW READ-ONLY TO ALL USERS WHO KNOW THE PASSWORD AND FINALLY TO ONLY USERS WHO HAVE THE SAME JOB NUMBER AND PROGRAMMER NAME AND THE PASSWORD. IN THE HIERARCHIC STRUCTURE, THERE IS A SIMPLE ALGORITHM TO DETERMINE THE LOCK-OUT MECHANISM, NAMELY AT EACH LEVEL OF LIBRARY STRUCTURING PROVIDE A FLAG TO INDICATE ITS RELATIVE PROTECTION. OTHER TECHNIQUES ARE DEVELOPABLE, BUT THE PREVIOUS METHOD IS THE SIMPLEST.

FOR PRIVATE LIBRARIES, A SIMPLE MECHANISM FOR SECURITY IS TO SPECIFY THE DDNAME TO BE THE SAME NAME OF A PASSWORD PROCESSOR, A LOAD MODULE RESIDING IN UHTSS/2 LOADLIB. WHEN THE USER SPECIFIES THIS DDNAME FOR THE UHTSS/2 ALLOCATOR, THE ALLOCATOR WILL USE THAT DDNAME AS THE ENTRY POINT NAME AND ATTACH IT FROM LOADLIB. THE USER PASSWORD PROCESSOR CAN THEN INTERROGATE THE USER AND RETURN HIS STATUS. HE IS EITHER ALLOWED TO USE THE DATA SET ENTIRELY, OR ONLY SOME MEMBERS, OR NOT AT ALL, OR THE SYSTEM SHOULD INVESTIGATE IMMEDIATELY THAT SOMEONE IS ATTEMPTING TO BREAK INTO HIS LIBRARY.

PASSWORD SECURITY SHOULD BE ENFORCED AT THE ALLOCATION OF RESOURCES WHICH SHOULD OCCUR ONLY AT THE INITIALIZATION OF THE JOB. THE USER WOULD NOT ENJOY CONSTANT QUERY OF HIS ABILITY TO USE THE RESOURCES.

APPENDIX:

USER PROGRAMS EXIST IN ONE OF THREE FORMS:

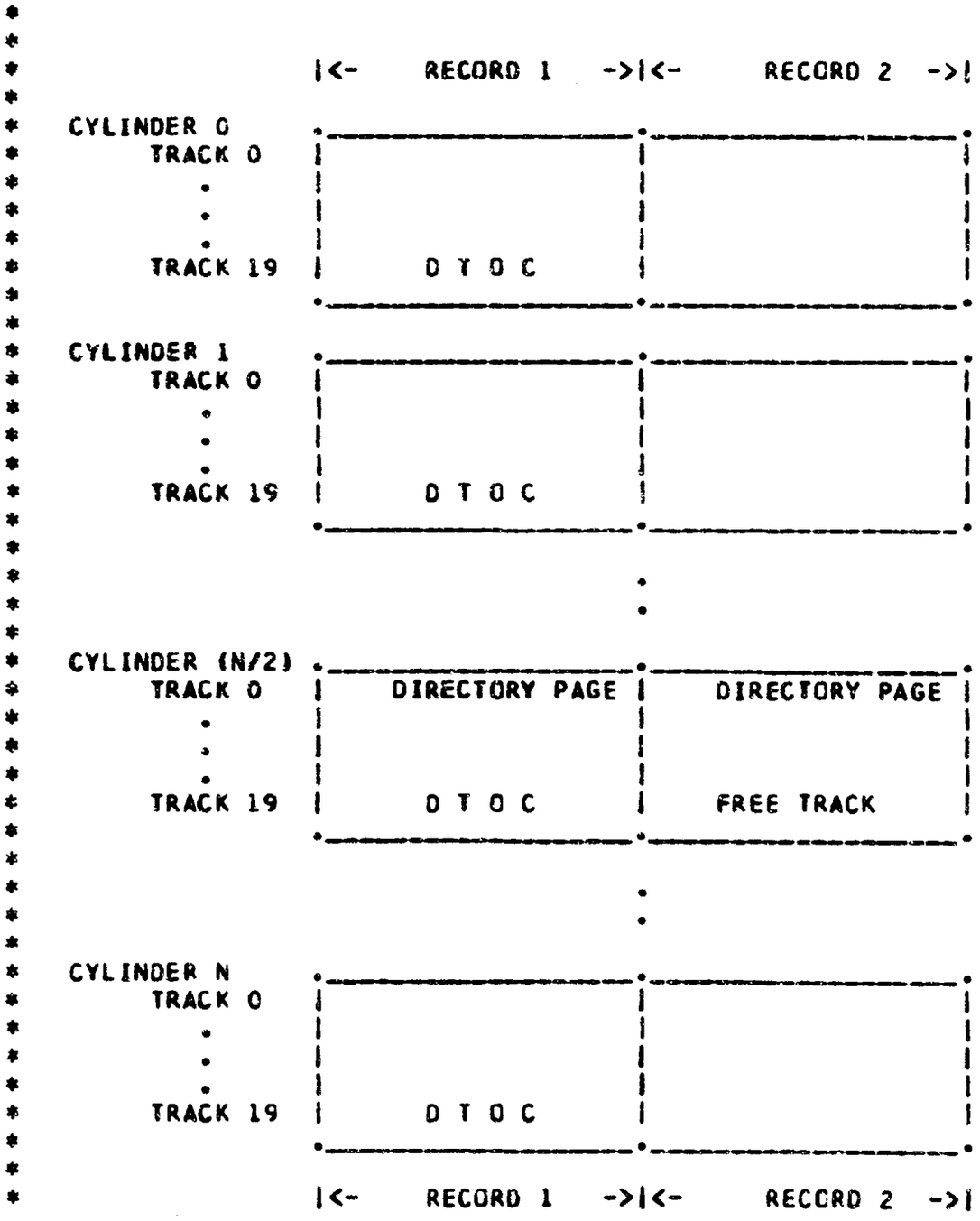
1. SOURCE FORM IS THE CARD-IMAGE INPUT TO HIGH-LEVEL LANGUAGES SUCH AS FORTRAN AND ASSEMBLER.

2. OBJECT FORM IS THE CARD-IMAGE OUTPUT OF HIGH-LEVEL LANGUAGES, I.E. RELOCATABLE BINARY FORM.

3. LOAD FORM IS THE OUTPUT OF THE LINKAGE EDITOR, I.E. A COMPACTED RELOCATABLE BINARY FORM.

OBJECT OR LOAD MODULES (PROGRAMS IN OBJECT OR LOAD FORM) ARE USED TO AVOID RECOMPILATION AND ARE GENERALLY DEBUGGED ROUTINES. THE USE OF THESE MODULES WILL DECREASE THE LINKAGE EDITOR'S TIME. LOAD MODULES ARE THE MOST DESIRED FORM BECAUSE THEY ARE LINKED FASTER, BUT SUFFER THE DISADVANTAGE OF REQUIRING DIRECT ACCESS ALLOCATION.

STRUCTURE OF OS/MVT IMPLEMENTATION



DESCRIPTION OF OS/MVT IMPLEMENTATION

UHTSS USERLIB IS A COLLECTION OF USER LIBRARIES (DATA SETS) FOR THE UHTSS PROJECT. ITS ORGANIZATION IS DIRECT, PARTITIONED, AND INDEXED SEQUENTIAL TO MINIMIZE DISK SEEK TIME, TO REDUCE DISK READ/WRITES, AND TO EFFICIENTLY UTILIZE DISK SPACE. IT IS DIRECT BECAUSE ANY MEMBER OF ANY USER LIBRARY IS ACCESSIBLE GIVEN ITS ADDRESS. IT IS PARTITIONED BECAUSE LIBRARIES ARE DIVIDED LOGICALLLY WITHIN THE USERLIB. FINALLY, IT IS INDEXED SEQUENTIAL BECAUSE ANY USER MEMBER IS SEQUENTIALLY ORGANIZED WITH A KEY.

THE PHYSICAL ORGANIZATION OF THE UHTSS USERLIB IS NEITHER LIMITED TO ONLY A SET OF CONTIGUOUS CYLINDERS NOR LIMITED TO A SINGLE DISK PACK. UHTSS USERLIB IS ORGANIZED AND SO STRUCTURED THAT IF IT RESIDES ON A DEDICATED DISK PACK, ARM CONTENTION AND SEEK TIMES WILL BE KEPT AT A MINIMUM. THE PRESENT IMPLEMENTATION OF USERLIB IS TO MAKE IT AN OS DIRECT DATA SET WITH HALF TRACK BLOCKS. EACH HALF TRACK BLOCK IS CALLED A LOGICAL TRACK, LT, AND IS CONSECUTIVELY NUMBERED. THE NUMBER OF LOGICAL TRACKS IS FORTY TIMES THE NUMBER OF ALLOCATED CYLINDERS. NOTE THAT ONLY THE PRIMARY EXTENT IS CONSIDERED. THE DISK ADDRESS OF EACH LOGICAL TRACK IS COMPUTABLE FROM THE LOGICAL TRACK NUMBER, LT#, WHERE CYLINDER=LT#/40, TRACK=MOD(LT#,40)/2, AND RECORD=MOD(LT#,2)+1. NOTE: HEREAFTER, DATA SETS WILL REFER TO USER DATA SETS, NOT THE USERLIB DATA SET.

THE MAJOR REASON FOR HALF TRACK RECORDS IS THAT THE UPDATE PROCESS WILL BE COMPLETED IN 1 1/2 REVOLUTIONS; 1/2 REVOLUTION TO READ IT IN, 1/2 REVOLUTION TO UPDATE IN CORE, AND 1/2 REVOLUTION TO WRITE IT OUT AGAIN. THIS IS ABOUT THE FASTEST METHOD OF UPDATING THE DISK GIVEN A SINGLE BUFFER. NOTE TOO, THAT THE HIGH FREQUENCY OF DISK ACTIVITY REQUIRES INFORMATION TO BE WRITTEN IMMEDIATELY IN ORDER TO INSURE RELIABILITY.

TIMING CONSIDERATIONS ON THE IBM 2314:

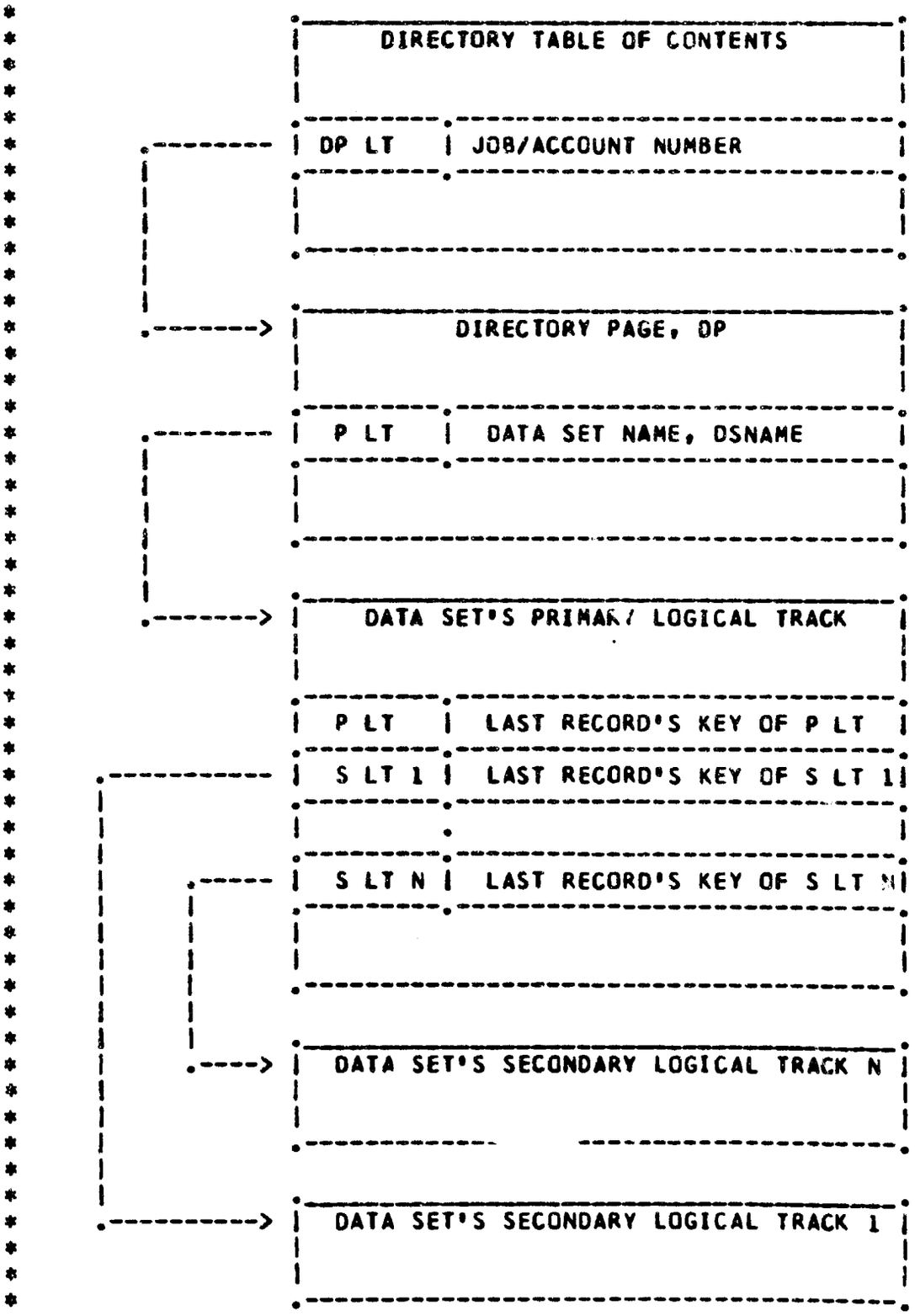
1) DATA TRANSFER OF 3520 BYTES IS 11.5 MS.

2) MINIMUM SEEK TIME TO AN ADJACENT CYLINDER IS 25 MS.

3) AVERAGE ROTATIONAL DELAY IS 12.5 MS.

4) DESIGN AVERAGE SEEK TIME IS 37.5 MS.

STRUCTURE OF UHISS USERLIB



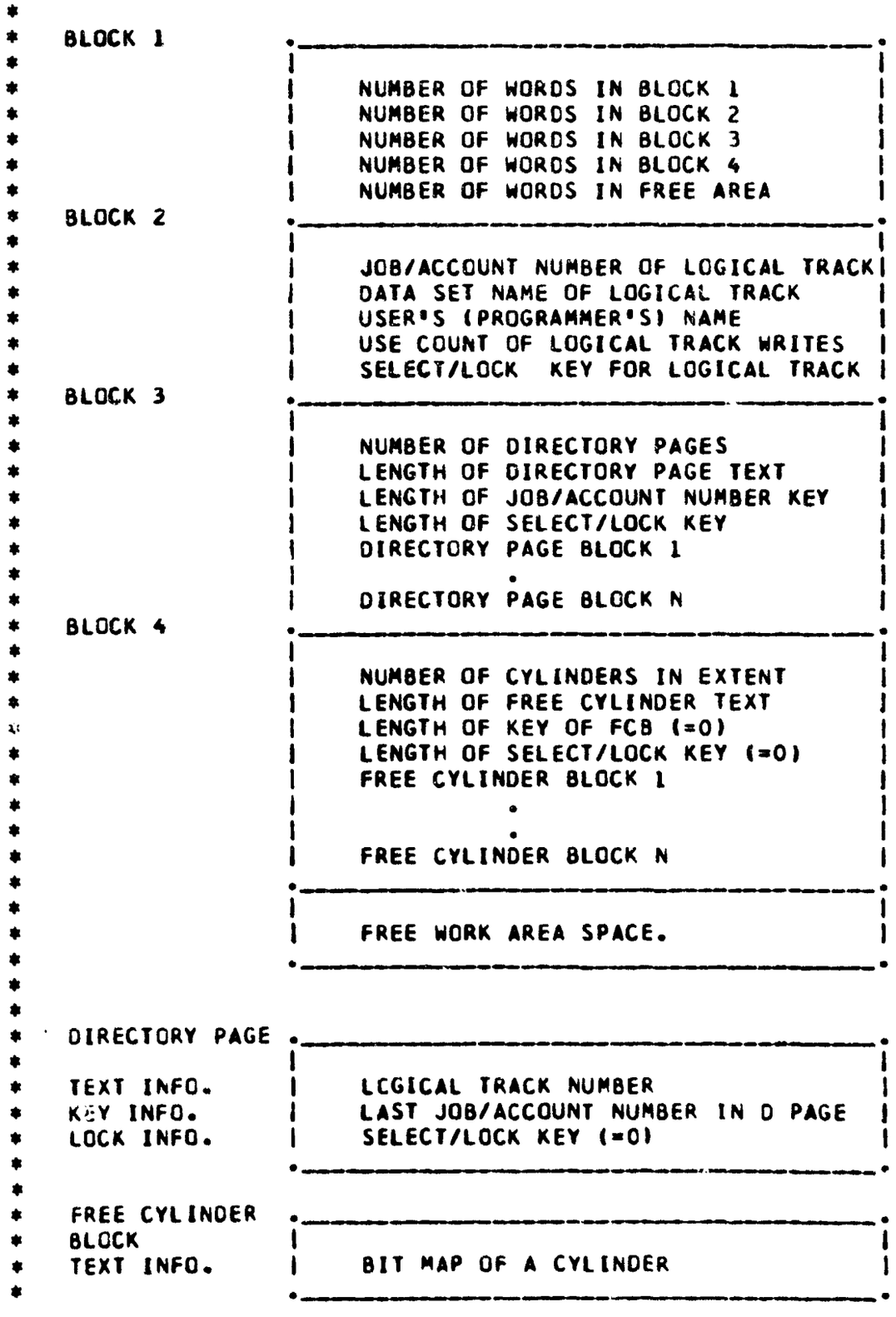
DESCRIPTION OF UHISS USERLIB

THE LOGICAL ORGANIZATION OF THE DATA SET CONSISTS OF A DIRECTORY AND USER MEMBERS (DATA SETS). THE DIRECTORY CONSISTS OF A DIRECTORY TABLE OF CONTENTS, DTOC, AND SEVERAL DIRECTORY PAGES, DP. THE KEY TO ANY USER DATA SET IS ITS JOB NUMBER, ACCOUNT NUMBER, AND DATA SET NAME, DSNAME. THE DTOC PROVIDES THE DP NUMBER GIVEN THE KEY CONSISTING OF THE JOB AND ACCOUNT NUMBERS. THE DP PROVIDES THE LT# OF THE DATA SET GIVEN THE KEY CONSISTING OF THE JOB AND ACCOUNT NUMBERS AND THE DATA SET NAME. A DATA SET CONSISTS OF A PRIMARY LOGICAL TRACK AND PERHAPS SEVERAL SECONDARY LOGICAL TRACKS. THE PRIMARY LT CONTAINS A LIST OF POINTERS TO THE OTHER SECONDARY LTS. THE SECONDARY LTS ARE NOT CHAINED IN ORDER TO MINIMIZE THE UPDATING PROCESS. WITHIN THE PRIMARY LT AND THE SECONDARY LTS, THE INDIVIDUAL RECORDS (CARD IMAGES) ARE SEQUENTIAL WITH RESPECT TO THEIR ASSOCIATED KEYS.

THE TREE STRUCTURE OF "USERLIB" IS INDEXED SEQUENTIAL WITH 3 LEVELS OF DIRECTORIES. THE FIRST DIRECTORY, WHICH IS CORE RESIDENT, PARTITIONS THE DATA SETS BY JOB/ACCOUNT NUMBER INTO PAGES. THE SECOND DIRECTORY, RESIDING ON DISK, PARTITIONS BY DATA SET NAME BY JOB/ACCOUNT NUMBER INTO THE ACTUAL DATA SETS. THE LAST DIRECTORY, CONTAINED WITHIN THE DATA SET ITSELF ON DISK, PARTITIONS THE DATA SET INTO LOGICAL TRACKS.

THUS TO RETRIEVE A BLOCK OF INFORMATION FROM ANY DATA SET INVOLVES: FIRST, AN IN-CORE SEARCH OF THE DTOC; NEXT, A DISK READ AND SEARCH OF THE DIRECTORY PAGE, A DISK READ AND SEARCH OF THE PRIMARY TRACK; AND FINALLY, A DISK READ OF THE TRACK THAT HAS THE BLOCK OF INFORMATION. NOTE THAT IF THE TRACK THAT HAS THE DESIRED BLOCK OF INFORMATION IS THE PRIMARY TRACK, THE TOTAL NUMBER OF DISK READS IS TWO.

STRUCTURE OF DIRECTORY TABLE OF CONTENTS, DIOC



DESCRIPTION OF DIRECTORY TABLE OF CONTENTS, DTOC

THE DTOC PARTITIONS THE USER DATA SETS SO THAT THE DIRECTORY PAGES ARE EVENLY ALLOCATED. USERS OF THE SAME JOB NUMBER WITH MANY DATA SETS ARE ALLOCATED AN ENTIRE DIRECTORY PAGE. THE PARTITIONING ALGORITHM IS A MANAGEMENT DECISION SCHEME.

THE DTOC CONTROLS THE USE OF THE FREE LOGICAL TRACKS, FLT, TO EFFICIENTLY ALLOCATE AND GARBAGE COLLECT. EACH ALLOCATION/COLLECTION CYCLE UPDATES THE DTOC AND THUS REQUIRES THE REWRITING OF THE NEW DTOC ONTO DISK TO INSURE SYSTEMS RELIABILITY.

EACH CYLINDER HAS A VERSION OF THE DTOC AS ITS FIRST RECORD OF THE LAST TRACK. BEFORE EACH REWRITE, THE DTOC USE COUNT FIELD IS INCREMENTED BY ONE. THE UPDATED DTOC IS THEN REWRITTEN ONTO THE CYLINDER WHERE THE ARM IS TO BE POSITIONED FOR THE NEXT READ/WRITE. THE DTOC WITH THE HIGHEST USE COUNT IS THE CURRENT DTOC. THE CURRENT DTOC IS MODE CORE-RESIDENT DURING THE UHTSS INITIALIZATION PROCESS.

THE CREATION OF A DATA SET WILL CAUSE AN ALLOCATION OF ONLY THE PRIMARY LT. SECONDARY LTS ARE ALLOCATED ONLY WHEN THEY ARE REQUESTED. IN THIS MANNER THE USERLIB WILL MINIMIZE THE DEAD SPACE/GAS IN EACH USER'S DATA SET.

THE DIRECTORY PAGE BLOCK IS A DESCRIPTOR RECORD WHICH CONTAINS A POINTER TO THE PROPER PAGE TRACK. THIS POINTER IS THE PAGE LOGICAL TRACK NUMBER.

AT THE PRESENT TIME THE DTOCS REQUIRE 2.5% OF THE DATA SET CAPACITY.

STRUCTURE OF DIRECTORY PAGE, DP

```
*   BLOCK 1          .-------------------------------------.
*                    |                                     |
*                    |   NUMBER OF WORDS IN BLOCK 1        |
*                    |   NUMBER OF WORDS IN BLOCK 2        |
*                    |   NUMBER OF WORDS IN BLOCK 3        |
*                    |   NUMBER OF WORDS IN BLOCK 4        |
*                    |   NUMBER OF WORDS IN FREE AREA      |
*   BLOCK 2          .-------------------------------------.
*                    |                                     |
*                    |   JOB/ACCOUNT NUMBER OF LOGICAL TRACK|
*                    |   DATA SET NAME OF LOGICAL TRACK    |
*                    |   USER'S (PROGRAMMER'S) NAME        |
*                    |   USE COUNT OF LOGICAL TRACK WRITES |
*                    |   SELECT/LOCK  KEY FOR LOGICAL TRACK|
*   BLOCK 3          .-------------------------------------.
*                    |                                     |
*                    |   NUMBER OF ASSOCATED DIRECTORY PAGES|
*                    |   LENGTH OF DIRECTORY PAGE TEXT     |
*                    |   LENGTH OF DATA SET NAME KEY       |
*                    |   LENGTH OF SELECT/LOCK KEY (=0)    |
*                    |   DIRECTORY PAGE BLOCK 1            |
*                    |   .                                 |
*                    |   .                                 |
*                    |   DIRECTORY PAGE BLOCK N            |
*   BLOCK 4          .-------------------------------------.
*                    |                                     |
*                    |   NUMBER OF DATA SETS IN DP         |
*                    |   LENGTH OF DATA SET TEXT           |
*                    |   LENGTH OF DATA SET KEY            |
*                    |   LENGTH OF SELECT/LOCK KEY         |
*                    |   DATA SET DIRECTORY BLOCK 1        |
*                    |            .                        |
*                    |            .                        |
*                    |   DATA SET DIRECTORY BLOCK N        |
*                    .-------------------------------------.
*                    |                                     |
*                    |   FREE WORK AREA SPACE              |
*                    .-------------------------------------.
*
*   DATA SET BLOCK   .-------------------------------------.
*                    |                                     |
*   TEXT INFO.       |   LOGICAL TRACK NUMBER OF PRIMARY LT|
*                    |   CREATION DATE OF DATA SET         |
*                    |   EXPIRATION DATE OF DATA SET       |
*                    |   LAST USED DATE OF DATA SET        |
*                    |   USE COUNT OF DATA SET             |
*                    |   USER'S (PROGRAMMER'S) NAME        |
*   KEY. INFO        |   JOB/ACCOUNT NUMBER OF DATA SET    |
*                    |   DATA SET NAME                     |
*   LOCK INFO.       |   DATA SET SELECTION/LOCKOUT KEY    |
*                    .-------------------------------------.
```

DESCRIPTION OF DIRECTORY PAGE, DP

THE DP CONTAINS A SEQUENTIAL LIST OF USER DATA SETS ORDERED BY JOB NUMBER, ACCOUNT NUMBER, AND DSNAME. THE LIST HAS THE LT# OF THE PRIMARY LT AND ACCOUNTING/MAINTENANCE INFORMATION. THE "LAST USED" DATE AND USE COUNT FIELD REFLECTS THE FREQUENCY OF USE OF THE DATA SET. THE USER WILL BE CHARGED ACCORDINGLY AND THE UHTSS MANAGEMENT CAN DETERMINE WHETHER OR NOT THE DATA SET IS TO BE MOVED TO THE LEAST ACCESSED REGION OF USERLIB OR EVEN ONTO DEAD STORAGE (TAPE).

THE PHYSICAL LOCATION OF THE DP IS THE CENTER CYLINDER OF USERLIB. THE CENTRAL CYLINDER STRATEGY MINIMIZES THE SEEK TIME FOR USER DATA SETS SINCE THE DP IS ALWAYS ACCESSED FIRST. THUS THE AVERAGE SEEK TIME BECOMES 37.5 MS FOR A DEDICATED PACK.

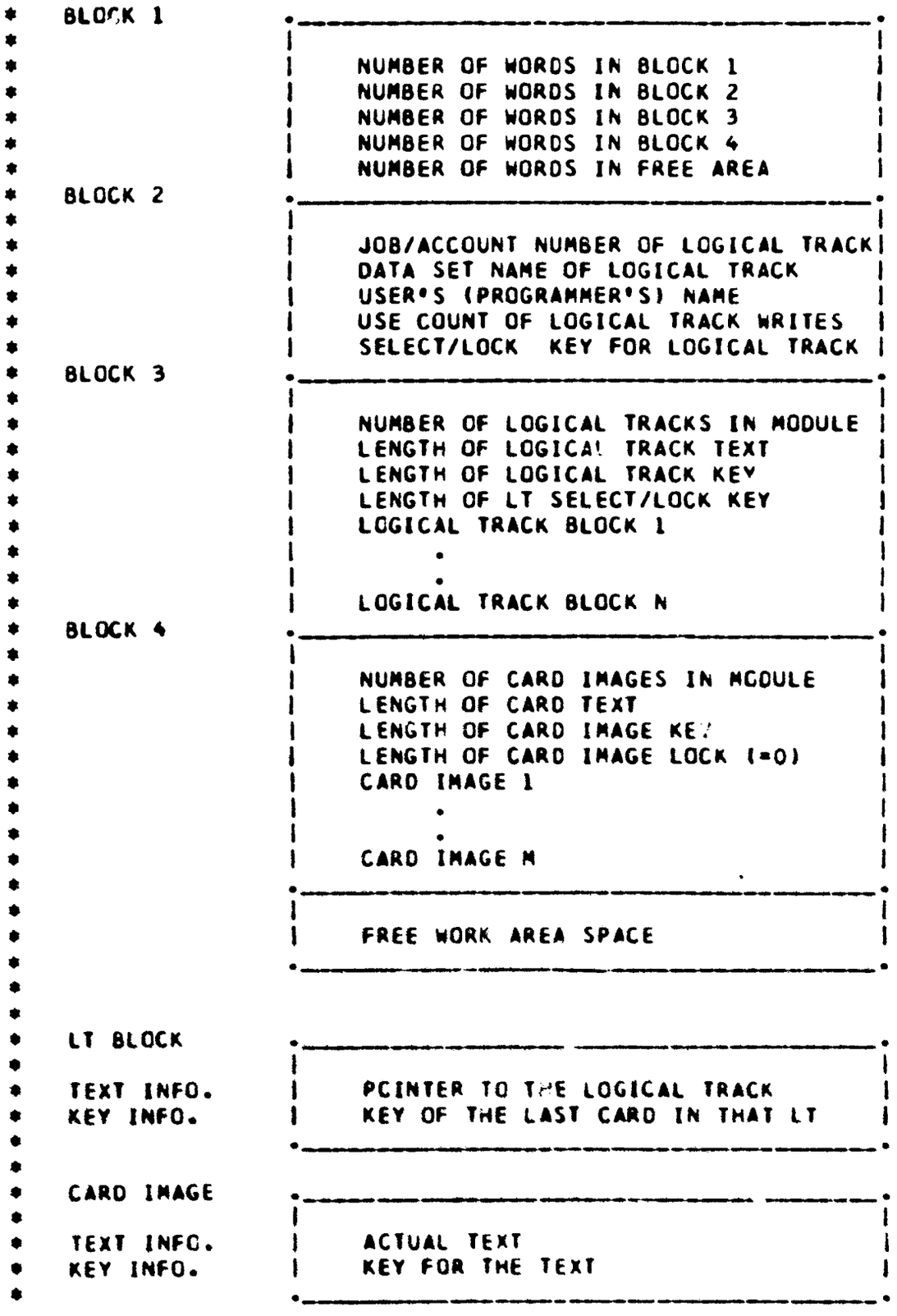
THE SELECTION OF A USER DATA SET REQUIRES ONLY ONE DISK READ. FROM THE CORE RESIDENT DTOC, THE LT OF THE DP ASSOCIATED WITH THE DSNAME IS OBTAINED. THAT LOGICAL TRACK IS THEN READ. FROM THE DP, THE LT# OF THE PRIMARY LT OF THE DATA SET IS OBTAINED. NOTE THAT THE DATA SET KEY IS THE JOB/ACCOUNT NUMBER CONCATENATED WITH THE DATA SET NAME.

THE DATA SET DIRECTORY BLOCK IS A DESCRIPTOR RECORD WHICH HAS A POINTER TO THE PRIMARY TRACK OF THE DATA SET. THE DATES IN THE RECORD ALLOW THE MANAGEMENT ROUTINES TO DETERMINE THE STATUS OF THE DATA SET.

AT THE PRESENT TIME, THE DIRECTORY PAGES CONSIST OF 0.9% OF THE DATA SET CAPACITY, ABOUT 1 CYLINDER OUT OF 101 CYLINDERS.

THE ALLOCATION OF JOB/ACCOUNTS TO PAGE TRACKS DEPENDS UPON THE FREQUENCY USE BY THOSE USERS. AT THE PRESENT IT IS ARBITRARILY ASSIGNED. AN ALGORITHM FOR PAGE OVERFLOW MUST BE DEVISED. ONE POSSIBLE STRATEGY IS TO MAKE DATA SET BLOCKS INTO CARD IMAGES AND USE THE TRACK OVERFLOW MECHANISM. BUT THERE ARE PROBLEMS IN POSITIONING THE PAGE TRACKS NEAR THE CENTRAL CYLINDER. THE ASSOCIATED DIRECTORY PAGE BLOCKS WILL BE USED FOR THE OVERFLOW ALGORITHM.

STRUCTURE OF UHISS DATA SET



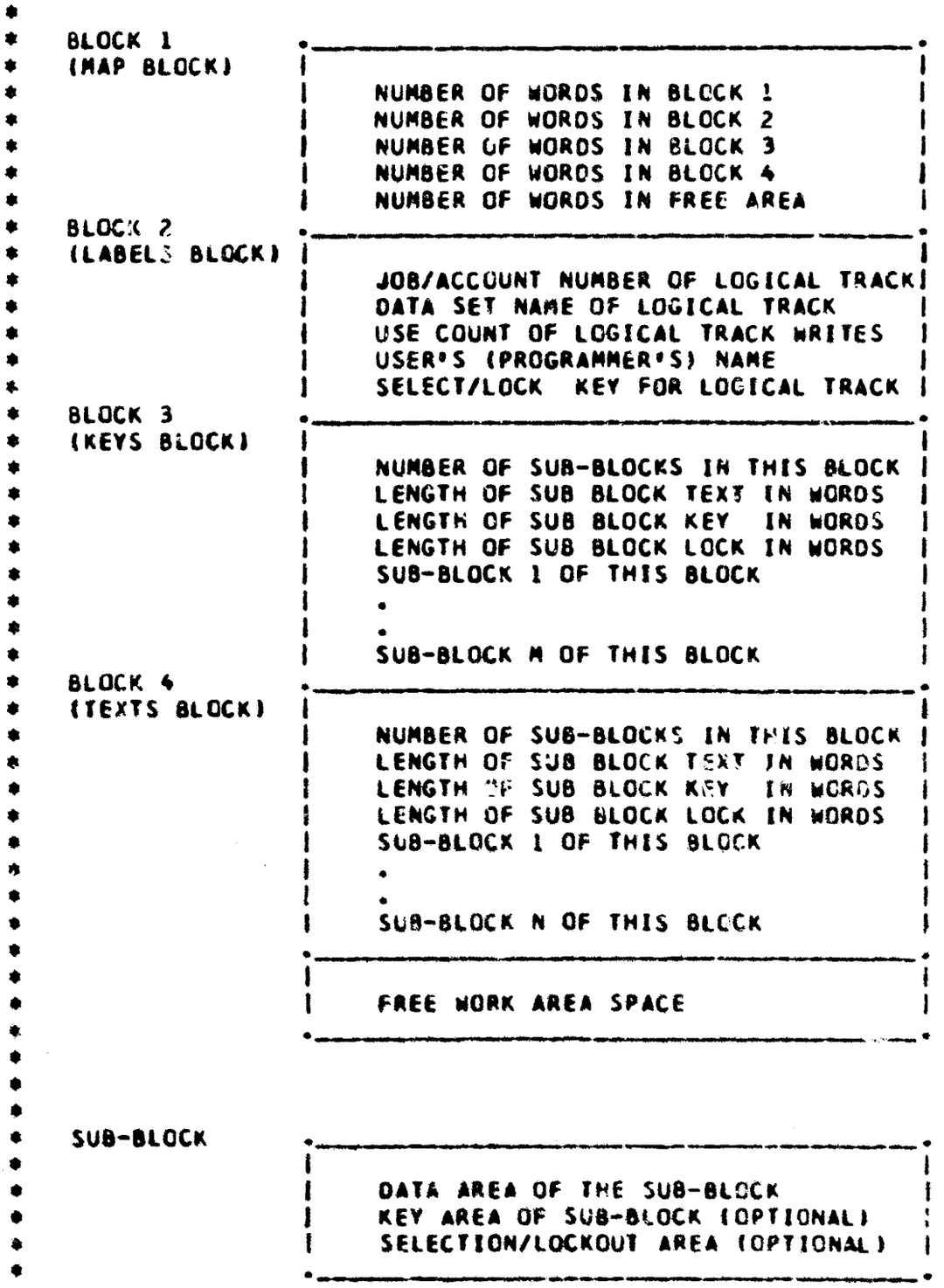
DESCRIPTION OF UHISS DATA SET

THE USER DATA SET CONSISTS OF A PRIMARY LT AND PERHAPS SEVERAL NON-CONSECUTIVE SECONDARY LTS. THE PRIMARY LT HAS A LIST OF SECONDARY LTS THAT IS ORDERED BY THE KEY OF THE LAST RECORD OF THE LT. A SEGMENT OF THIS LIST IS RESIDENT WITHIN THE ACTIVE TERMINAL TABLE TO MINIMIZE THE READING OF THE PRIMARY LT FOR THE LT ASSOCIATED WITH THE UPDATE KEY. EACH LT CONTAINS CARD IMAGE RECORDS ORDERED BY THEIR ASSOCIATED KEY.

THE UPDATE OF A USER DATA SET REQUIRES ONLY 1 DISK REVOLUTION FROM THE INITIAL READ. FROM THE ACTIVE TERMINAL TABLE, THE LT# OF THE ASSOCIATED LT IS OBTAINED. THE PROCESS OF READING THE LT, UPDATING IT, AND WRITING IT OCCURS WITHIN 1 1/2 REVOLUTIONS BECAUSE THE UPDATE PROCESS TAKES LESS THAN A HALF REVOLUTION. FOR UPDATES WITHIN A DENSE REGION OF THE USER DATA SET, THE MAXIMUM DISK READ/WRITES IS 2. FOR SPARSE UPDATING , THE MAXIMUM NUMBER OF DISK READ/WRITES IS 3.

THE PRIMARY USE OF THE USER DATA SET IS TO STORE USER PROGRAMS IN CARD IMAGE FORM. HOWEVER, A SECONDARY FEATURE PERMITS XPL BINARY PROGRAM IMAGES.

## APPENDIX GENERALIZED STRUCTURE FOR LOGICAL TRACKS



DESCRIPTION OF APPENDIX

THE MAP BLOCK PROVIDES A MAP OF THE LOGICAL TRACK, I.E. WHERE EACH BLOCK IS LOCATED. THUS THE RELATIVE ADDRESS OF BLOCK 4 IS THE ADDRESS OF BLOCK 1 PLUS THE SUM OF THE NUMBER OF BYTES IN BLOCKS 1, 2 AND 3. USING THIS MAPPING TECHNIQUE, INDIVIDUAL BLOCKS MAY BE EXPANDED OR CONTRACTED WITHOUT AFFECTING THE OTHER BLOCK ADDRESSES.

THE LABELS BLOCK PROVIDES INFORMATION ABOUT THE LOGICAL TRACK ITSELF. IT ASSOCIATES THE LOGICAL TRACK TO A JOB/ACCOUNT NUMBER AND A DATA SET NAME. FOR SYSTEM LOGICAL TRACKS, THE DATA SET NAMES ARE "DTCC" OR "D PAGE". IT ALSO PROVIDES INFORMATION REGARDING THE NUMBER OF TIMES THE LOGICAL TRACK HAS BEEN WRITTEN.

THE KEYS BLOCK PROVIDES A LIST OF KEYS AND ASSOCIATED POINTERS FOR DATA CHAINING. THE POINTERS ARE LOGICAL TRACK NUMBERS.

THE TEXTS BLOCK PROVIDES A SEQUENTIAL FILE OF KEYED RECORDS. IF THERE IS A REQUEST FOR AN UNKEYED FILE, THE LENGTH OF THE KEY AREA SHOULD BE ZERO. NOTE THAT BOTH THE KEYS AND TEXTS BLOCKS HAVE THE SAME STRUCTURE. THE SAME STRUCTURING ALLOWS ONLY 1 PROGRAM TO PERFORM THE DATA MANAGEMENT. THE SELECTION/LOCKOUT AREA IS PROVIDED TO INSURE A LEVEL OF PROTECTION FOR INDIVIDUAL RECORDS. SINCE THE SELECTION/LOCKOUT AREA IS OPTIONAL, THE LENGTH OF THE SELECT/LOCK KEY MAY BE ZERO.

IMPORTANT NOTE

THIS LOGICAL TRACK HAS BEEN ORGANIZED IN WORDS EVEN THOUGH THERE IS AN OBVIOUS INEFFICIENCY IN STORAGE USAGE. IT IS MORE IMPORTANT TO GAIN SPEED IN COMPUTATION THAN MINIMIZING STORAGE.

PHYSICAL CHARACTERISTICS OF THE DISK FILE ON THE IBM 2314 UNIT:

USING A STAND-ALONE ROUTINE, DASD PRINT, THE FOLLOWING INFORMATION ABOUT THE DISK FILE HAS BEEN VERIFIED: EACH PHYSICAL TRACK IS DIVIDED INTO 3 RECORDS. RECORD 0 IS A 16 BYTE RECORD WHERE THE FIRST 8 BYTES ARE FOR THE COUNT AND THE OTHER 8 FOR THE DATA. RECORDS 1 AND 2 ARE 3528 BYTE RECORDS WHERE THE FIRST 8 BYTES ARE FOR THE COUNT AND THE REMAINING 3520 BYTES FOR THE DATA. THE COUNT AREA IDENTIFIES THE RECORD (IN TERMS OF CYLINDER NUMBER, HEAD NUMBER, AND RECORD NUMBER) AND INDICATES THE RECORD'S FORMAT (COUNT-DATA) AND LENGTH. THE DATA AREA CONTAINS THE RECORD INFORMATION.

EXTENSIONS TO THE LIBRARY

IF MORE THAN ONE DISK PACK IS AVAILABLE, IT IS EASY TO TREAT THE LIBRARY AS ONE GARGANTUAN DISK PACK WITH A TABLE TO TRANSLATE THE LT#S TO THE APPROPRIATE DISK PACKS. THIS SCHEME IS EQUIVALENT TO SOFTWARE PAGING TECHNIQUES.

IF MORE CORE IS AVAILABLE, THEN THE DISK READS SHOULD BE BUFFERED. ANY OPERATION UPON THE TRACK WOULD, HOWEVER, REQUIRE IMMEDIATE DISK WRITES. AGAIN THIS IS TO INCREASE RELIABILITY.

THE CARD IMAGES MAY BE "PRESSED-DECK" TO SOME DEGREE. ESSENTIALLY THE TRAILING BLANKS COULD BE DELETED. THIS EXTENSION WILL INCREASE THE STORAGE CAPACITY OF THE DISK FILE AND NOT SIGNIFICANTLY DECREASE THE USUAL PROCESSING. THE MAJOR DESIRE FOR SUCH AN IMPLEMENTATION IS TO DECREASE THE NUMBER OF CHARACTERS TO BE TYPED BY THE TELETYPES. TO THIS END, THE KEY BLOCK SHOULD PRECEDE THE TEXT BLOCK.

IT IS NOW CLEAR THAT THE USE OF POINTERS WITHIN THE TRACK IS FEASIBLE IF IT WOULD DECREASE DATA MOVEMENT AND THE ADDED EXPENSE OF 2 BYTES PER RECORD IS NOT PROHIBITIVE. IN FACT ONE COULD PROBABLY KEEP VARIABLE LENGTH RECORDS WITH POINTERS. AT TRACK OVERFLOW TIME, A GARBAGE COLLECTOR WOULD BE EMPLOYED. HOWEVER, TO IMPLEMENT SUCH AN INITIAL DESIGN IS TOO COMPLICATED A TASK CONSIDERING THE MAGNITUDE OF THE OVERALL STRUCTURE OF THE LIBRARY MANAGEMENT. IT IS CERTAINLY EASIER TO DEVELOP THE EDIT AND DATA MANAGEMENT ROUTINES FIRST KNOWING THAT ONE IS PLAYING WITH CARD IMAGES.

# TRACK MANAGEMENT

## ALLOCATION OF LOGICAL TRACKS

THE FREE LOGICAL TRACKS ARE KEPT IN A POOL IN ORDER TO ALLOCATE THEM IN THE MOST EFFICIENT MANNER. SINCE THE ALLOCATION SCHEME POSITIONS THE TRACKS NEAR THE CENTRAL CYLINDER, ALL THE DATA SETS WILL BE CLUSTERED ABOUT THE CENTRAL CYLINDER. THEN THE AVERAGE SEEK TIME WILL BE NEAR MINIMUM.

1. EXAMINE THE CORE TABLE'S FREE CYLINDER BLOCK.
2. FIND A CYLINDER NEAR TO THE CENTRAL CYLINDER.
3. EXAMINE THE BIT MAP OF THE FREE LTS IN THE CYLINDER BLOCK.
4. TAKE THE FIRST FREE LT AND INDICATE THAT IT'S ALLOCATED.
5. SIGNAL THAT THE DTOC MUST BE REWRITTEN.

## COLLECTION OF FREE LOGICAL TRACKS

THE COLLECTION OF FREE TRACKS USUALLY OCCURS DURING THE LOGICAL TRACK UNDERFLOW CONDITION. HOWEVER UTILITY PROGRAM WILL ALSO PERFORM THIS GARBAGE COLLECTION TO ELIMINATE UNUSED OR EXPIRED DATA SETS.

0. EXAMINE THE CORE TABLE'S FREE CYLINDER BLOCK.
1. DETERMINE THE CYLINDER NUMBER FROM THE LT NUMBER.
2. DETERMINE THE TRACK NUMBER FROM THE LT NUMBER.
3. DETERMINE THE RECORD NUMBER FROM THE LT NUMBER.
4. GET THE CYLINDER BLOCK CORRESPONDING TO THE COMPUTED CYLINDER NUMBER.
5. SET THE APPROPRIATE BIT TO INDICATE THAT IT IS FREE.
7. INDICATE THAT THE DTOC MUST BE REWRITTEN.

## LOGICAL TRACK OVERFLOW

WHEN AN INSERTED CARD FILLS THE LOGICAL TRACK TO CAPACITY, A LOGICAL TRACK OVERFLOW CONDITION EXISTS. THE LT MUST THEN BE REORGANIZED TO ALLOW FURTHER EXPANSION. THIS REORGANIZATION REQUIRES ONLY 4 DISK READ/WRITES. THE PROCESS REQUIRES THE FOLLOWING STEPS.

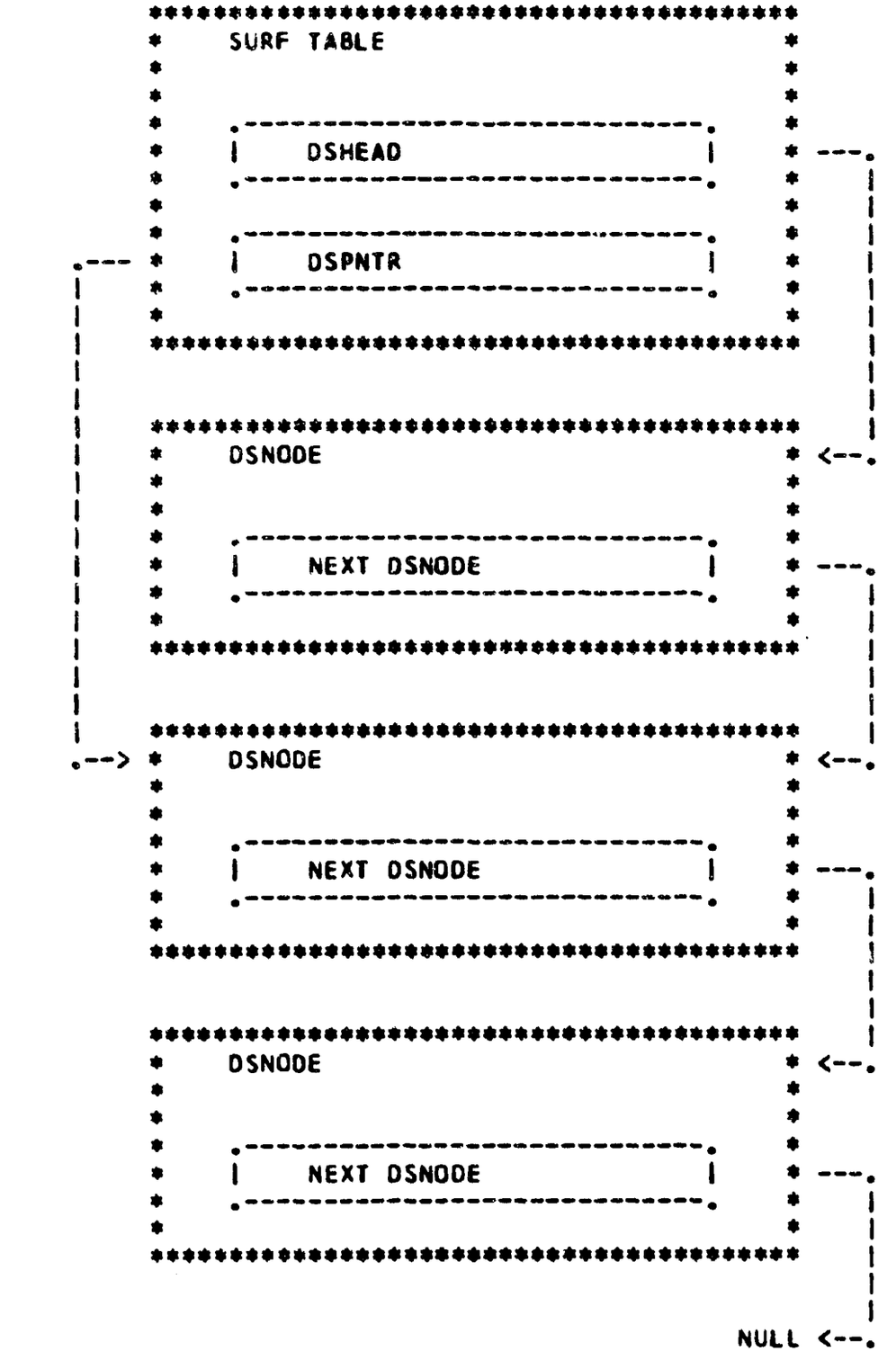
1. SAVE THE LAST 15 CARDS OF THE FILLED TRACK.
2. WRITE OUT THE UPDATED TRACK.
3. GET THE PRIMARY LOGICAL TRACK.
4. GET A FREE LOGICAL TRACK.
5. UPDATE THE CHAINING INFORMATION.
6. WRITE OUT THE PLT.
7. CREATE A NEW FLT IMAGE IN THE BUFFER.
8. INSERT THE SAVED 15 CARDS.
9. WRITE OUT THE NEW TRACK.

## LOGICAL TRACK UNDERFLOW

WHEN THE RESULT OF A DELETION REQUEST LEAVES ONE OR NO CARD IN A TRACK, A LOGICAL TRACK UNDERFLOW CONDITION EXISTS. IF THE TRACK IS EMPTY, IT IS SIMPLY FREED BACK TO THE SYSTEM AND REMOVED FOR THE PRIMARY TRACK DIRECTORY. FOR THE OTHER CASE (ONLY A CARD LEFT), EITHER THE FIRST OR THE LAST CARD IS INSERTED INTO THE NEXT LOGICAL TRACK AFTER THE TRACK IS FREED TO THE SYSTEM.

## DATA SET MANAGEMENT

### MULTI DATA SET ALLOCATION



## MULTI DATA SET ALLOCATION

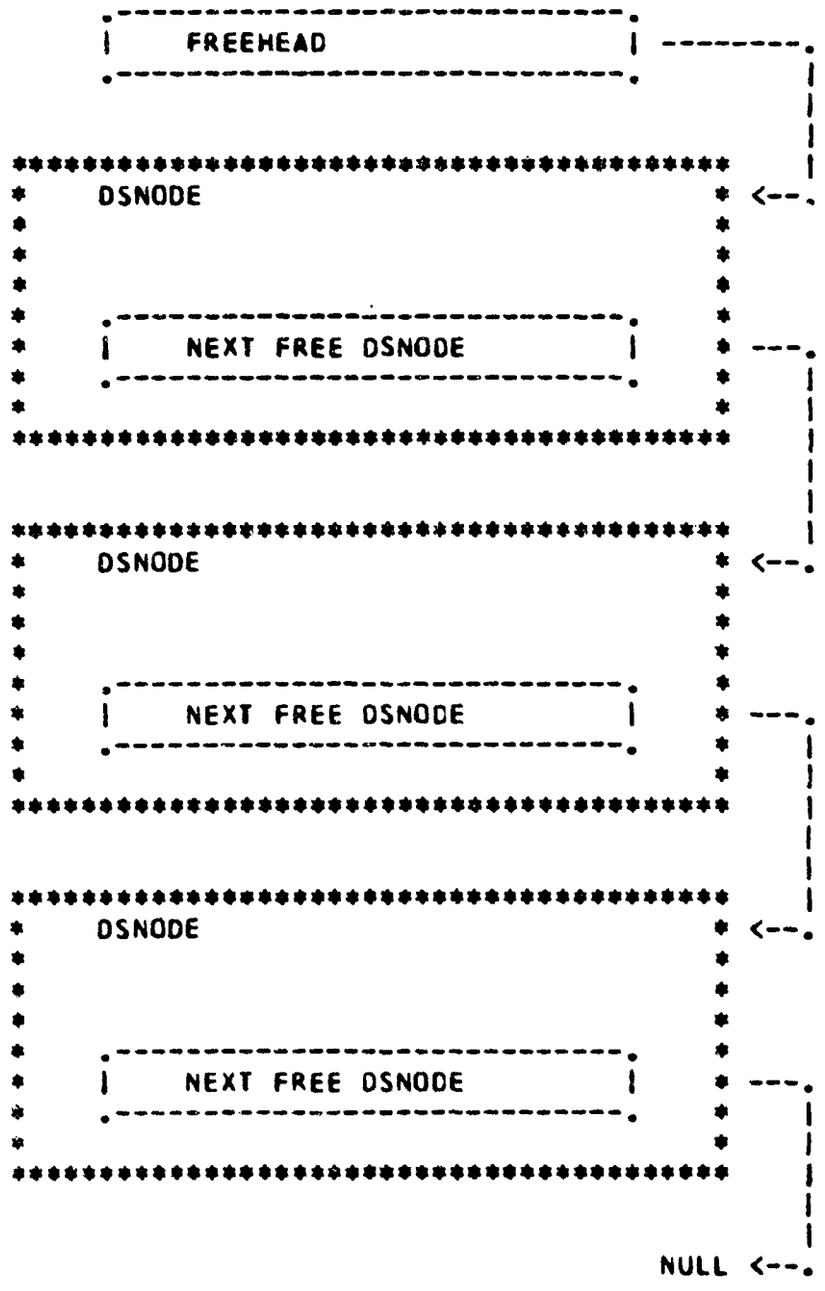
THE MULTI DATA SET ALLOCATION SCHEME PROVIDES THE SWAPPING PROGRAM THE ABILITY TO ACCESS MORE THAN ONE DATA SET. IT ACCOMPLISHES THIS FUNCTION BY ALLOCATING A DATA SET CONTROL BLOCK, DSNODE, FOR FOR EACH DATA SET. THE DSNODES CONTAIN THE INFORMATION REQUIRED TO PROCESS A DATA SET, NAMELY THE DATA SET NAME, PRIMARY LOGICAL TRACK NUMBER, THE SECONDARY LOGICAL TRACK NUMBER, AND THE APPROPRIATE KEYS.

AT INITIALIZATION, A LIST OF FREE DSNODES IS CREATED WITH FREEHEAD AS THE ACCESS POINTER. THEN EACH TERMINAL IN THE SYSTEM IS ALLOCATED A DSNODE.

WHENEVER A "SELECT" OR "CREATE" COMMAND IS ISSUED FROM "SMON", A NEW DSNODE IS OBTAINED FROM THE FREE DSNODE LIST AND STACKED ONTO THAT TERMINAL'S LIST. EACH TERMINAL HAS A POINTER TO THE HEAD OF DSNODE STACK , DSHEAD, AND A POINTER TO THE CURRENT DSNODE, DSPNTR. WHEN THE DATA SETS ARE NO LONGER REQUIRED, ALL DSNODES EXCEPT THE LAST ONE IN THE STACK IS REMOVED AND RETURNED TO THE FREE DSNODE LIST.

MULTI DATA SET ALLOCATION

FREE DSNODE LIST



## SELECTION PROCESS

THE SELECTION PROCESS ALLOWS THE USER TO ACCESS THE UHTSS LIBRARY. THE APPROPRIATE ENTRIES IN THE UHTSS TABLES WILL INDICATE THAT FACT.

1. EXAMINE THE CORE TABLE'S DIRECTORY PAGE BLOCK.
2. USING THE JOB/ACCOUNT NUMBER, OBTAIN THE ASSOCIATED DP ENTRY.
3. FROM THE ASSOCIATED DP ENTRY, OBTAIN THE LT# OF THE DP.
4. READ IN THE ASSOCIATED DIRECTORY PAGE.
5. USING THE JOB/ACCOUNT NUMBER AND THE DATA SET NAME, OBTAIN THE ASSOCIATED DATA SET BLOCK.
6. FROM THE ASSOCIATED DATA SET BLOCK, OBTAIN THE PRIMARY LT# OF THE DATA SET.
7. FILL THE UHTSS TABLES WITH THE APPROPRIATE INFORMATION.
8. UPDATE THE CURRENT DATE FIELD AND WRITE IT BACK OUT.
9. ISSUE WARNING IF CURRENT DATE IS GREATER THAN EXPIRATION DATE.
10. POSITION THE DATA SET TO THE FIRST RECORD ON THE DATA SET. (SKIP THIS IF THE DATA SET IS LOADABLE.)

## CREATION PROCESS

THE CREATION PROCESS INVOLVES THE CREATION OF A FREE TRACK IMAGE IN THE BUFFER AND WRITING IT OUT. THE DIRECTORY PAGE IS ALSO UPDATED.

## SCRATCH PROCESS

SCRATCHING WRITES A FREE TRACK IMAGE ONTO DISK FOR EACH LT TRACK. THEN THE ENTRY IN THE DIRECTORY PAGE IS DELETED.

## LOADABLE DATA SET

A LOADABLE DATA SET, LDS, IS AN USER DATA SET IN THE UHTSS LIBRARY WHICH CONTAINS THE XPL GENERATED CORE IMAGE OF A LOAD MODULE. EACH TRACK OF THE DATA SET CONTAINS 3000 BYTES OF THE CORE IMAGE. THE FIRST TRACK CONTAINS A CONTROL BLOCK OF 60 BYTES.

THE GENERATION OF A LDS REQUIRES THAT THE COMPILER GENERATE THE ENTIRE DATA SET IN CORE WITH THE CONTROL BLOCK. THE CODE MUST BE RELOCATABLE. THE COMPILER THEN ISSUES THE LINK COMMAND, WHICH SELECTS A DATA SET. UPON SUCCESSFUL COMPLETION OF THE COMMAND, THE "DAOL" COMMAND WITH A KEY IS ISSUED TO WRITE OUT THE DATA SET.

THE FUNCTION OF "DAOL" IS TO WRITE OUT BLOCKS OF CORE IMAGES ONTO DISK. THE USER MUST GIVE A KEY IN THE COMMAND FOR THAT BLOCK OF CORE CORRESPONDING TO "AREALOC". NOTE THAT NOT ONLY CORE IMAGES BUT BLOCKED CARD IMAGES MAY BE WRITTEN. "DAOL" ACTS EXACTLY LIKE "INSERT", NAMELY ADDS A NEW LOGICAL TRACK IF THE KEY DOES NOT EXIST AND REPLACES IT OTHERWISE.

## ACCESSING DATA SETS

IN ORDER FOR THE USER PROGRAM TO READ A DATA SET, HE MUST FIRST DECIDE HOW HE IS GOING TO ACCESS THE RECORDS. CURRENTLY, DATA SETS ARE READ ONLY FOR BOTH COMPILER INPUT AND INPUTTING COMPILERS. THUS THE DATA SET MUST BE SELECTED IN ORDER TO OBTAIN THE PRIMARY LOGICAL TRACK ADDRESS. SINCE INPUTTING COMPILERS AND COMPILER INPUT ARE SPECIAL KINDS OF REQUESTS THE LINK COMMAND PERFORMS THE SELECT AND STUFFS THE SURF BUFFER WITH SOME CONTROL INFORMATION. THE MOST INFORMATION IN THE SURF BUFFER IS MEANINGLESS FOR COMPILER INPUT WHILE SOME OF THE DATA IS MEANINGFUL FOR INPUTTING COMPILERS. THE CONTROL INFORMATION IS DISCUSSED IN THE CONTROL RECORD FORMAT SECTION.

THE MOST EFFICIENT METHOD OF READING IN A DATA SET IS TO READ IN LOGICAL TRACKS. FOR INPUTTING COMPILERS, A LOGICAL TRACK IS 3000 BYTE OF CODE. BUT FOR COMPILER INPUT, A LOGICAL TRACK CONTAINS SEVERAL CARD IMAGES, I.E. IT IS ESSENTIALLY A VARIABLE NUMBER OF CARD IMAGES (80 BYTES). THUS THE TARGET AREA SPECIFIED IN AREALOC MUST BE 3536 BYTES, 3520 FOR THE BLOCKED RECORDS AND 16 BYTES FOR THE BLOCK CONTROL INFORMATION (NUMBER OF CARDS IN THE TRACK AND OTHER DATA). TO ADD SOME FLEXIBILITY THE LOAD COMMAND MUST INCLUDE THE INDEX, WHERE INDEX TELLS WHICH LOGICAL TRACK IS TO BE READ. NOTE THAT INDEX RANGES FROM 0 TO #SUBBLOCKS-1. THUS TO READ IN A COMPILER, ONE MUST ISSUE AS MANY LOADS AS THERE ARE LOGICAL TRACKS. AREALOC MUST BE INCREMENTED EVERY TIME. TO READ IN COMPILER INPUT, THERE IS AN ADDITIONAL COMPLEXITY, I.E. THE USER MUST DEBLOCK THE RECORDS USING THE BLOCK CONTROL INFORMATION.

## CONTROL RECORD FORMAT

THE CONTROL BLOCK HAS THE FOLLOWING FORMAT:

| WORD | CONTENTS |
|---|---|
| 0 | NUMBER OF BYTES OF FULL PROGRAM CODE |
| 1 | NUMBER OF BYTES OF FULL DATA CODE |
| 2 | NUMBER OF PROGRAM RECORDS |
| 3 | NUMBER OF DATA RECORDS |
| 4 | NUMBER OF BYTES IN A RECORD |
| 5 | NUMBER OF BYTES IN THE LAST PROGRAM RECORD |
| 6 | NUMBER OF BYTES IN THE LAST DATA RECORD |
| 8 | NUMBER OF LOGICAL TRACKS IN THE DATASET. (AUGMENTED ONLY DURING THE LINK COMMAND) |
| 9 | NUMBER OF BYTES OF PROGRAM CODE |

WORDS 0 TO 6 ARE GENERATED BY THE XPL COMPILER AND ARE NOT MANIPULATED BY ISS-LIBRARY. WORD 8 IS STUFFED INTO THE SURF BUFFER ONLY FOR THE LINK COMMAND. IT DOES NOT EXIST ON THE DATA SET. WORD 9 IS GENERATED BY ISS-LIBRARY AND GIVES THE RELATIVE ADDRESS OF THE START OF THE DATA AREA FOR USE BY SMON.

## JOB MANAGEMENT

### TSS-LIBRARY QUEUE PROCESSING

TSS-LIBRARY QUEUE MAY HAVE MORE THAN ONE REQUEST AT ANY TIME. EACH REQUEST MAY REQUIRE EITHER SINGLE SERVICE OR MULTIPLE SERVICE. THE NEW DESIGN PHILOSOPHY IS TO PROCESS ALL REQUESTS AS SINGLE SERVICE IN A ROUND ROBIN FASHION. THE REASONS FOR THIS DESIGN IS PRIMARILY TO GIVE EQUITABLE TREATMENT TO REQUESTS (AND THEREFORE THE USERS) AND SECONDARILY TO GIVE SOME PSYCHOLOGICAL REASSURANCE TO THE USER.

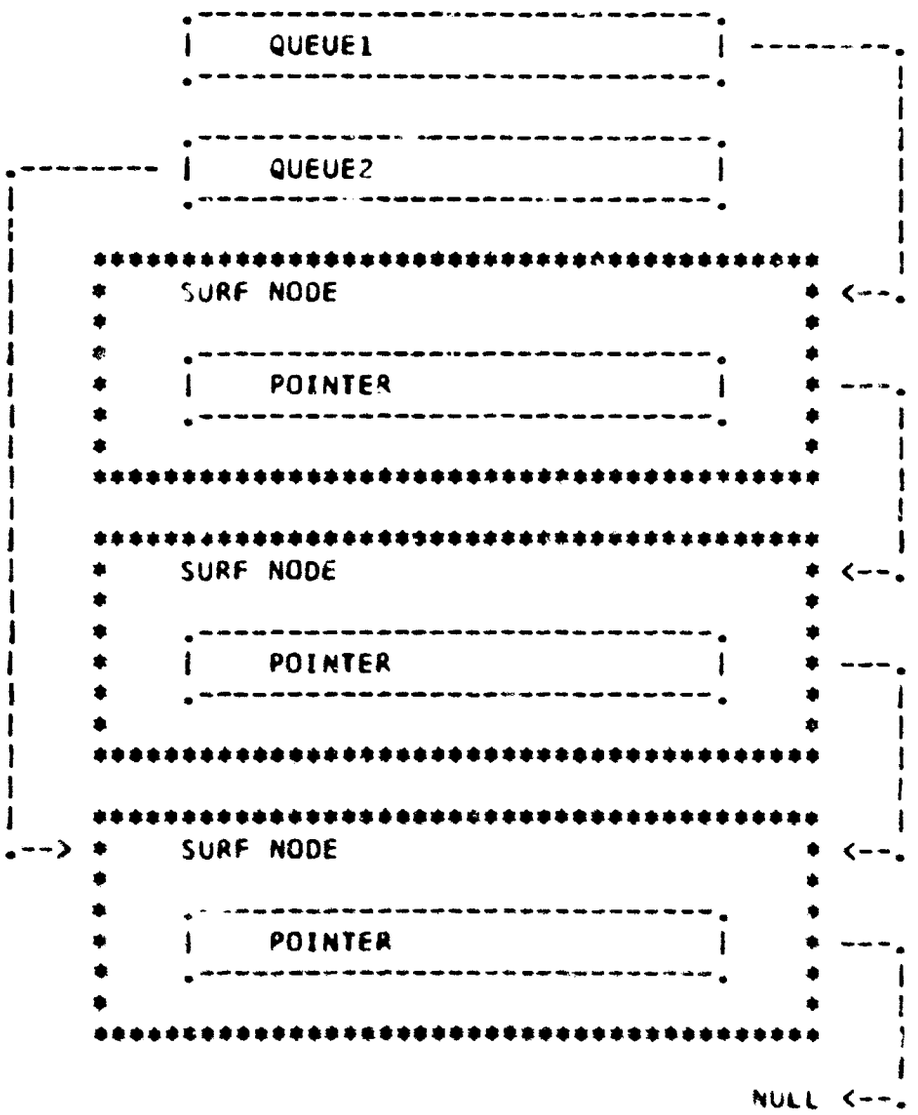
THIS ROUND ROBIN APPROACH IS THE SAME AS TIME SLICES IN THE LARGER SYSTEM ENVIRONMENT. THUS A USER WHOSE REQUEST IS QUEUED AFTER FIVE OTHERS WILL RECEIVE SOME SERVICE AFTER PROCESSING THE FIFTH SERVICE REQUEST. THIS APPROACH DIFFERS RADICALLY FROM THE INITIAL DESIGN WHICH WAS A FIFO QUEUE WHERE THERE WAS NO DISTINCTION BETWEEN SINGLE SERVICE AND MULTIPLE SERVICE REQUESTS. FOR SINGLE SERVICE REQUESTS FIFO OR ROUND ROBIN PROCESSING IS EQUIVALENT. HOWEVER, FOR MULTIPLE SERVICE REQUESTS, THE "ROUND ROBIN" APPROACH LETS THE USER KNOW EVERY SO OFTEN (DEPENDS ON THE QUEUE LENGTH) THAT HIS REQUEST IS BEING SERVICED, THUS REASSURING HIM THAT HE IS STILL ACTIVE WITHIN THE SYSTEM.

THERE IS STILL S AN INHERENT INEFFICIENCY IN THE ROUND ROBIN APPROACH WITHIN THE TSS-LIBRARY ENVIRONMENT. THIS ENVIRONMENT MUST NOW READ IN THE PRIMARY LOGICAL TRACK TO DETERMINE WHICH SECONDARY LOGICAL TRACK IS TO BE OPERATED UPON. THUS THERE ARE ADDITIONAL DISK READS EVERY TIME A REQUEST IS SERVICED. THE FIFO SCHEME COULD SAVE THE NECESSARY INFORMATION AND THUS ELIMINATE MOST OF THE READING OF THE PRIMARY TRACKS.

THE IMPLEMENTATION OF MULTIPLE SERVICE ROUTINES REQUIRES A NEW EXAMINATION OF THE USUAL COMMAND STRUCTURE. ESSENTIALLY A FIELD, COUNT, MUST KEEP A RECORD OF THE NUMBER OF TIMES THE COMMAND IS TO BE EXECUTED. ADDITIONAL FIELDS MUST CONTAIN THE PROPER INFORMATION TO SERVE AS INDICES FOR VARIOUS PARAMETERS. NOTE THAT THE COMMAND NEED NOT BE WITHIN THE BUFFER SINCE LIBTYPE ALREADY INDICATES THE COMMAND. THUS IF COUNT IS ZERO, THE REQUEST IS SINGLE SERVICE, OTHERWISE IT IS MULTIPLE SERVICE REQUEST. EVENTUALLY WHEN "COUNT" BECOMES ZERO, THE REQUEST WILL BE DELETED FROM THE QUEUE. FOR COMPLEX OPERATIONS IN "OUTPUTER" SUBIOTYP IS USED TO DISTINGUISH THE SUB-FUNCTIONS.

FINALLY, THERE IS ANOTHER COMPLICATION, I.E. THE HASP SUBMISSION. THE ROUND ROBIN APPROACH CANNOT BE USED SINCE IT WOULD INTERSPERSE ONE USER'S JCL WITH ANOTHER'S JCL. FOR THIS CASE, ONLY ONE USER IS PROCESSED WHILE THE OTHER HASP REQUESTS ARE PASSED OVER. THUS WITHIN THE ROUND ROBIN STRUCTURE THERE IS A SINGLE SERVER SUBQUEUE.

## QUEUE PROCESSING




TMON PLACES REQUESTS IN A QUEUE FOR TSS-LIBRARY. WHEN TSS-LIBRARY HAS SATISFIED THE REQUEST, IT POSTS THE COMPLETION OF ITS TASK AND TMON REMOVES THE ELEMENT FROM THE QUEUE. "QUEUE1" POINTS TO THE FIRST REQUEST WHILE "QUEUE2" TO THE LAST. IN THE SURF TABLE FOR A TERMINAL THERE IS A POINTER TO THE NEXT REQUEST IN THE TSS-LIBRARY QUEUE. WHEN THE POINTER IS NEGATIVE, TSS-LIBRARY HAS REACHED THE END OF THE QUEUE AND PROCEEDS TO REPROCESS THE QUEUE FROM THE BEGINNING TILL ALL REQUESTS HAVE BEEN SATISFIED. WHEN ALL REQUESTS HAVE BEEN SATISFIED, TSS-LIBRARY WILL WAIT UNTIL TMON GIVES IT MORE WORK.

## BATCHING PROCESSING

THE NUMBER OF SERVICES UHTSS OFFERS IS LIMITED. IN ORDER TO UTILIZE THE FULL CAPABILITY OF THE OPERATING SYSTEM, UHTSS PROVIDES A METHOD OF SUBMITTING JOBS TO THE BATCH PROCESSOR VIA THE TERMINALS.

THE HASP SYSTEM PROVIDES AN INTERNAL READER WHICH IS ALLOCATED USING A DD CARD ESSENTIALLY IT IS A PSUEDO CARD READER BEHAVING TO HASP AS AN ORDINARY CARD READER.

THE HASP COMMAND ROUTES THE CARDS TO THE INTERNAL READER. THESE CARDS HOWEVER MUST HAVE THE PROPER JCL CARDS (JUST LIKE A NORMAL JOB SUBMITTED TO THE CARD READER). BOTH THE PRINT AND PUNCH COMMANDS GENERATE THE PROPER JCL FOR THE USER USING THE SYSTEM DEFAULTS. NOTE THAT ALL '/*' CARDS ARE REPLACED BY '**' CARDS.

## USING THE BATCH VERSION OF THE ISS LIBRARY PROGRAM

ALL COMMANDS FOR THE SYSTEM ARE PUNCHED ON CARDS IN EXACTLY THE SAME FORMAT AS ON THE CONSOLES. ALL REPLIES BY THE SYSTEM ARE PRINTED OUT. DISPLAY IS THE SAME AS PRINT. THE FORMAT OF THE POSTING IS THE TERMINAL NUMBER (ALWAYS 0), THE CURRENT DATA SET NAME (MAY BE BLANK), THE CURRENT TIME OF DAY IN BINARY IN TERMS OF HUNDREDTHS OF A SECOND, AND A MESSAGE. THE MESSAGE MAY BE THE COMMAND TO BE EXECUTED, THE RESPONSE OF THE SYSTEM (*:.....), A STATUS REPORT (SELECT: DATA SET SELECTED PROPERLY), AN ERROR MESSAGE (CREATE: DUPLICATE NAME ....), OR FINALLY THE REQUEST SATISFIED MESSAGE WITH THE CURRENT I/O COUNT.

NOTE: ONLY COMMANDS TO THE LIBRARY PROGRAM ARE RECOGNIZED. YOU CANNOT EXECUTE A PROGRAM BUT YOU MAY HASP A DATA SET. THE BATCH VERSION WILL TERMINATE ON ANY ERROR EXCEPT END OF DATA SET.

NOTE: THE COST OF USING THE BATCH VERSION IS COMPUTED AS A NORMAL BACKGROUND JOB. THE UHTSS/2 ACCOUNTING SCHEME DOES NOT APPLY. DEPENDING UPON THE APPLICATION THE REGION SIZE AND I/O COUNTS MAY BE SIGNIFICANT.

FIRST LINE PRINTED BY THE BATCH VERSION INFORMS YOU AS TO THE SIZE OF THE DYNAMIC AREA AVAILABLE TO DAOL/LOAD. IT MUST BE LARGER THAN YOUR XPL FORMATTED DATA SET. OTHERWISE THE PROGRAM TERMINATES. THIS DYNAMIC AREA IS A FUNCTION OF THE REGION SIZE OF THE STEP. THE BATCH VERSION TAKES ABOUT 90K TO EXECUTE. IF YOU WISH TO LOAD/LAOL, YOU MUST ADD THE APPROPRIATE AMOUNT TO 90K.

TO CREATE A LOADABLE DATA SET ON THE TIME SHARING FILE, THE FOLLOWING COMMANDS MUST BE EXECUTED:

1) HI ######## 'PROGRAMMER NAME'
(NOTE THAT ######## IS YOU JOB/ACCOUNT NUMBER. AT THE MOMENT THE SYSTEMS JOB/ACCOUNT NUMBER IS 00000000)

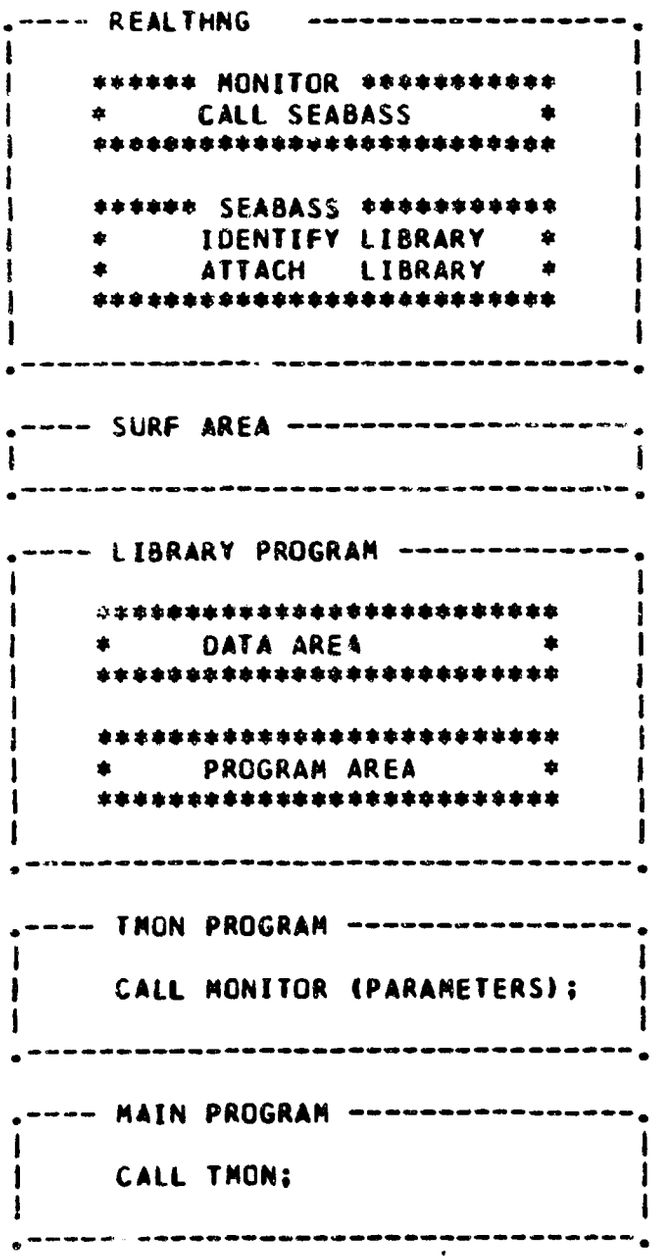
2) CREATE 'DATA SET NAME'
THE DATA SET NAME IS THE NAME OF THE PROGRAM YOU WANT TO EXECUTE.

3) DAOL 'DATA SET NAME'
YOUR LOADABLE DATA SET MUST BE IN XPL FORM ON INPUT2. THE DDCARD MUST REDEFINE THE DATA SET TO BE DCB=(RECFM=FB,LRECL=80,BLKSIZE=7200)

NOTE: THE APPROPRIATE MESSAGES WILL INFORM YOU OF THE COMPLETION OF EACH REQUEST AND THE PROPER EXECUTION OF EACH REQUEST.

## SOURCE LAYOUTS

### TMON/LIBRARY SOURCE LAYOUT




THE DIAGRAM DETAILS THE PROGRAM STRUCTURE OF TMON/LIBRARY. THE REAL THING IS A LOAD MODULE OF ASSEMBLY PROGRAMS. THE REST ARE WRITTEN IN XPL. THE SURF AREA IS A COLLECTION OF DECLARATIONS. SIMILARLY THE DATA AREA OF THE LIBRARY PROGRAM IS JUST DECLARATIONS. BOTH TMON AND LIBRARY ARE INTERNAL PROCEDURES TO THE MAIN PROGRAM WITH THE SURF AREA GLOBAL TO ALL THREE PROGRAMS.

## TMON/LIBRARY SOURCE LAYOUT

THE PROCESS OF VITALIZING THE LIBRARY PROGRAM HAS THE FOLLOWING STEPS:

A) THE MAIN PROGRAM CALLS TMON.

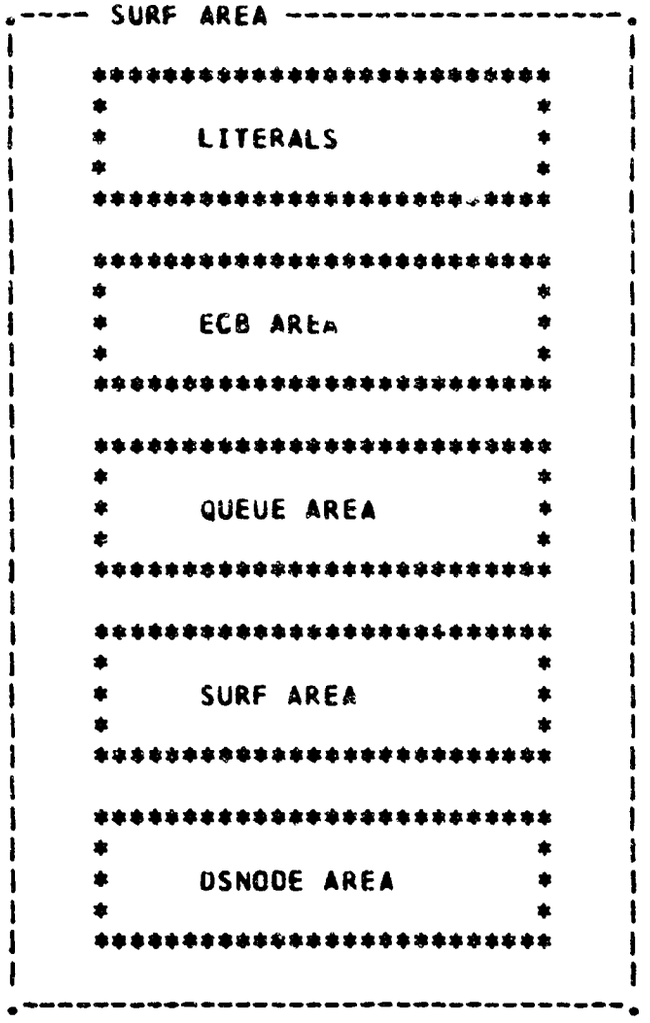
B) AFTER TMON INITIALIZES HIS AREAS, HE CALLS THE MONITOR AND ALSO PASSES A PARAMETER LIST (ONE SUCH PARAMETER IS THE ADDRESS OF TSS_LIBRARY).

C) THE MONITOR (AN ASSEMBLY PROGRAM) CALL SEABASS (ALSO IN ASSEMBLY).

D) SEABASS PROCEEDS TO "IDENTIFY" LIBRARY AND "ATTACHES" THE PROGRAM. THIS PROCESS MAKES LIBRARY KNOWN TO OS/MVT AS A TASK AND EXECUTES IT AS SUCH.

E) EVENTUALLY LIBRARY IS EXECUTED AS A TASK AND SPRINGS TO LIFE.

SURF AREA SOURCE LAYOUT




THE SURF AREA CONSISTS OF LITERALS, ECBS, QUEUES, SURF, AND THE DSNODE AREA. THESE DATA ELEMENTS ARE THE ONLY GLOBALS REQUIRED FOR COMMUNICATION BETWEEN TMON AND LIBRARY.

SURF AREA SOURCE LAYOUT

THE LITERALS ENABLE THE SYSTEM PROGRAMS TO REFER TO ELEMENTS OF SURF AND DSNODE IN A STRAIGHT-FORWARD MANNER. IT PARALLELS THE USE OF DSECTS OF ASSEMBLER LANGUAGE OR PREPROCESSOR STATEMENTS OF PL/1. NOTE THAT THE OBJECT CODE IS NOT OPTIMIZED BY THE USE OF SUCH MACROS.

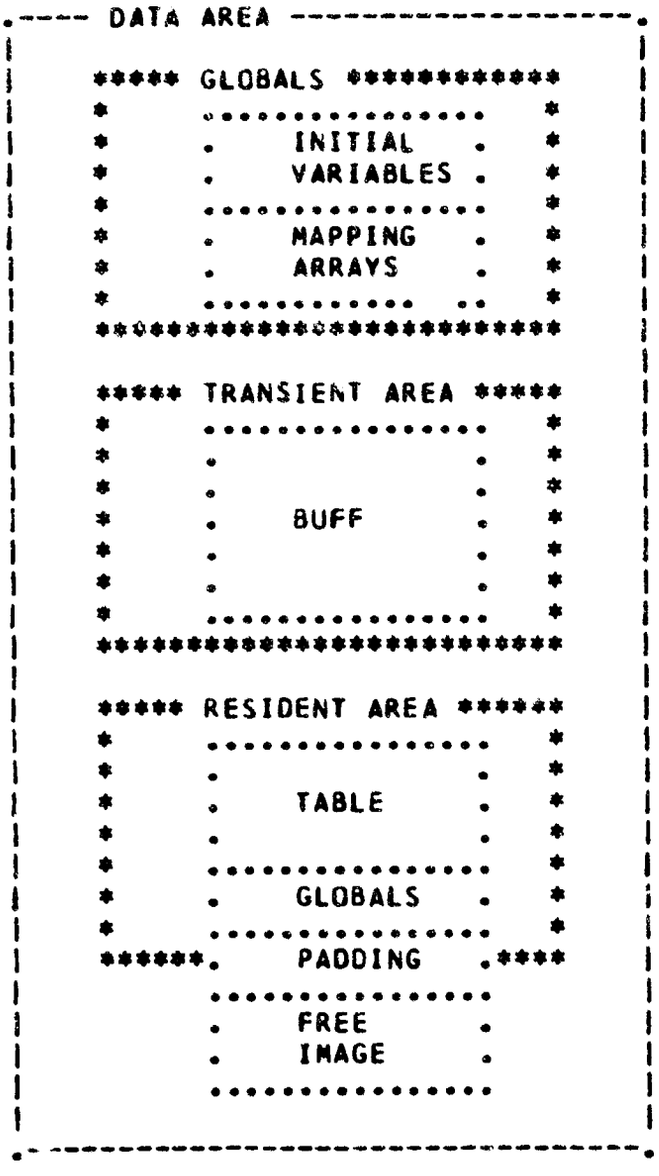
THE ECBS PROVIDE SYNCHRONIZATION OF TASKS (NAMELY TMON, LIBRARY, AND IOCONTRL). IN TIME THE NAMING CONVENTION WILL BE REGULARIZED TO REFLECT PROGRAMS, NOT PROGRAMMERS.

THERE ARE THREE QUEUES IN THE OVERALL SYSTEM. TMON MAINTAINS EACH ONE FOR THE SUBSIDIARY TASKS, IOCONTRL, SMON, AND LIBRARY. THESE QUEUES ARE JUST THE HEAD AND TAIL POINTERS TO SURF BLOCKS WHICH ARE LINKED IN THE APPROPRIATE MANNER.

SURF, THE SYSTEM USER'S FILE IS AN ARRAY OF CONTROL BLOCKS. EACH TERMINAL IS ASSIGNED A BLOCK WHICH CONTAINS ALL THE PERTINENT INFORMATION TO PROVIDE SERVICE TO THAT TERMINAL. THE SIZE HAS BEEN DECLARED TO BE 500 WORDS. HOWEVER THERE ARE CURRENTLY ONLY 6 TERMINALS DEFINED WITH 65 WORDS PER BLOCK. THUS SOME AREA OF CORE IS UNUSED. FROM AN AESTHETIC POINT OF VIEW, SURF BLOCKS SHOULD BE DYNAMIC, LIKE MAIN STORAGE TO HANDLE TRANSIENT TERMINALS.

ON THE OTHER HAND, DSNODE IS MORE DYNAMIC. PRESENTLY IT CAN HANDLE 40 DATA SET NODES(A DSNODE IS 20 WORDS AND THE AREA IS 800 WORDS). SURF CONTAINS POINTERS TO THE DSNODES (ACTUALLY ONLY INDICES). THERE IS A STACK ALLOCATION/COLLECTION MECHANISM THAT MANAGES THIS AREA.

LIBRARY DATA AREA SOURCE LAYOUT




THERE ARE FOUR MAJOR GLOBAL AREAS WITHIN THE LIBRARY PROGRAM. THE FIRST GLOBAL AREA CONSISTS OF VARIABLES WITH INITIAL VALUES AND A SET OF MAPPING ARRAYS. IN FACT IF THERE WERE MORE MACRO SPACE IN THE XPL COMPILER, MOST OF THE VARIABLES WOULD BE LITERALS. THE MAPPING ARRAYS CONTAIN STRUCTURAL INFORMATION REGARDING EITHER DISK BUFFER AREAS (BUFF OR TABLE BUT NOT BOTH).

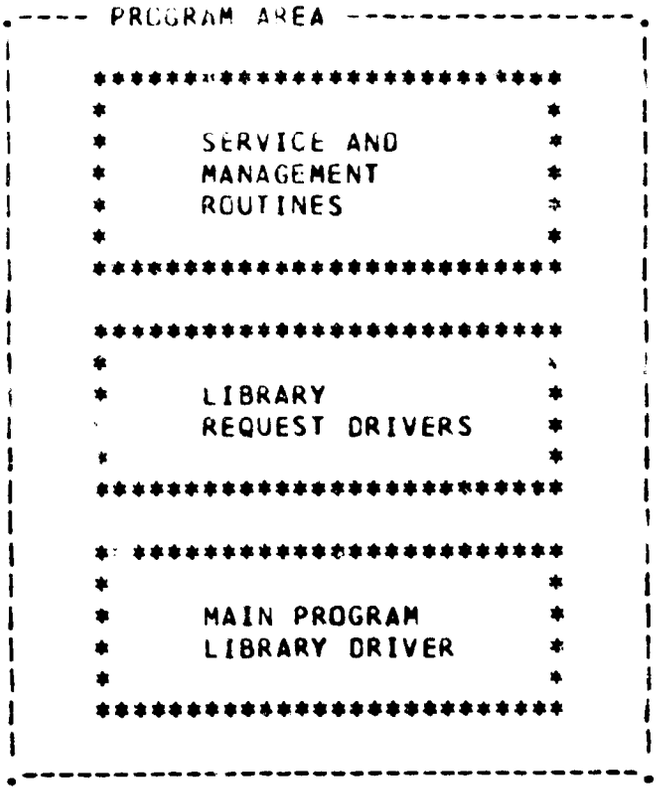
## LIBRARY DATA AREA SOURCE LAYOUT

THE SECOND GLOBAL AREA IS A TRANSIENT AREA, BUFF, FOR THE DISK OPERATIONS. CURRENTLY IT IS 3524 BYTES EVEN THOUGH THE RECORD SIZE IS 3520. THE EXTRA FOUR BYTES ARE FOR KICKS.

THE LAST TWO GLOBAL AREAS ARE OVERLAPPED. THE RESIDENT AREA, TABLE, IS INITIALIZED WITH THE CURRENT DTOC. AFTER INITIALIZATION THE AREA IS NEVER READ INTO AGAIN. OCCASIONALLY THE DTOC IS WRITTEN OUT, BUT NEVER READ IN THEREAFTER. CURRENTLY THE DTOC CONTAINS LESS THAN 400 WORDS BUT THE BUFFER AREA IS 880 WORDS. IN ORDER TO UTILIZE THIS SPACE, THE LAST GLOBAL AREA IS OVERLAYED ON TO THE FULL TABLE BUFFER. THIS AREA CONTAINS WORK AREAS, PADDING AND FREE_IMAGE.

THE WORK AREA CANNOT BE INITIALIZED SINCE IT IS OVERLAY DURING INITIALIZATION. PADDING IS AN AREA TO INSURE THAT THE DISK RECORD DOES NOT OVERRUN SOME OTHER AREA. FREE_IMAGE IS THE DISK IMAGE OF A FREE LOGICAL TRACK (IT IS CORE RESIDENT TO REDUCE DISK READS).

LIBRARY PROGRAM SOURCE LAYOUT




THE SUBROUTINE STRUCTURE IS SIMPLE SINCE XPL REQUIRES THAT PROCEDURES MUST EXIST BEFORE THEY ARE REFERENCED. THERE ARE VERY FEW INTERNAL PROCEDURES. MOST OF THE ROUTINES ARE CLUSTERED TO REFLECT SIMILAR FUNCTIONS; HOWEVER, THE ORDERING IS MAINLY HISTORICAL.

## SUBROUTINE COMPONENTS

### INITIALIZATION

THERE ARE THREE PHASES FOR INITIALIZATION. THE FIRST PHASE IS A ONE-SHOT EXECUTION, I.E. IT OCCURS ONLY AT THE START OF THE PROGRAM AND IS NEVER EXECUTED AGAIN. THE OTHER PHASES OCCUR DURING EVERY REQUEST FOR SERVICE.

AT THE FIRST CALL TO THE LIBRARY SYSTEM, THE ROUTINE "INITIALIZE" IS CALLED TO FIND THE CURRENT DTOC AND READ IT INTO THE CORE TABLE, SUPPLY THE PROPER ADDRESS REQUIRED THROUGHOUT THE PROGRAM, AND TO SAVE THE FREE LOGICAL TRACK IMAGE IN CORE FOR LATER USE. IT ALSO CREATES A FREE DSNODE LIST AND ALLOCATES A NODE TO EACH SURF TERMINAL.

AT EVERY REQUEST A LIST OF ADDRESSES MUST BE CALCULATED. ONE PHASE INITIALIZES POINTERS THAT DEPEND UPON "TERMAD" FOR SURF FIELDS. THE OTHER PHASE INITIALIZES POINTERS FOR THE DSNODES WHICH DEPEND UPON "DSPNTR".

### OVERLAYS

IN ORDER TO SAVE CORE STORAGE, VARIABLES NOT USED DURING INITIALIZATION OR VARIABLES NOT INITIALIZED ARE DECLARED AFTER "TABLE". "TABLE" IS THE BUFFER AREA FOR THE DTOC, WHICH MUST BE 3520 BYTES. BUT ONLY ABOUT 1600 BYTES IS USED FOR THE DTOC WHICH IS READ IN ONLY ONCE DURING INITIALIZATION. THE OTHER 1920 BYTES CAN BE EFFECTIVELY USED FOR OTHER VARIABLES. NOTE THAT "PADDING" IS AN ARRAY THAT ASSURES THAT THE AGGREGATE SIZE OF "TABLE", THE OTHER VARIABLES, AND "PADDING" IS GREATER THAN 3520.

COMMAND PROCESSING

THE MAJOR ROUTINE THAT DEALS WITH COMMAND PROCESSING IS "PARSING" WHICH DEALS WITH A COMMAND SYNTAX THAT FOLLOWS:

<COMMAND> := <COMMAND LIST> | <COMMAND LIST> ;

<COMMAND LIST> := <COMMAND HEAD>

<COMMAND LIST> := <COMMAND LIST> <DELIMITER> <COMMAND OPTION>

<COMMAND HEAD> := <COMMAND OPTION>

<COMMAND HEAD> := <NULL>

<COMMAND OPTION> := <POSITIONAL OPTION>

<COMMAND OPTION> := <KEYWORD OPTION>

<POSITIONAL OPTION> := <OPTION>

<KEYWORD OPTION> := <KEYWORD> = <OPTION>

<DELIMITER> := <BLANK> | <COMMA>

<OPTION> := <STRING OF CHARACTERS WITHOUT BLANKS>

<OPTION> := ' <STRING OF CHARACTERS WITH BLANKS> '

P.S. THE STANDARD PROBLEM OF APOSTROPHES OCCUR. TO REPRESENT THEM USE TWO SUCCESSIVE APOSTROPHES, I.E. ''.

IT IS IMPORTANT TO NOTE THAT IN THE COMMAND PROCESSING, THE FIRST CALL TO "PARSING" IS A DUMMY BECAUSE "TMON" HAS ALREADY DECODED THE COMMAND. THE FUNCTION OF THE FIRST CALL IS TO SET UP THE PROPER "IMAGE".

THE "PARSING" ROUTINE IS THE MOST IMPORTANT ROUTINE BECAUSE IT DOES ALL THE ANALYZING OF THE COMMAND SYNTAX. THE ROUTINE REQUIRES THAT THE GLOBAL VARIABLE "IMAGE" BE FILLED WITH THE COMMAND STRING, "BUFFER". AT EACH CALL TO "PARSING", TWO ANSWERS ARE RETURNED VIA GLOBALS "KEYWORD" AND "OPTION". THE ROUTINE IS CALLED AS MANY TIMES AS REQUIRED TO PROCESS ALL THE OPTIONS. NOTE THAT THE COMMAND ITSELF MUST BE VIEWED AS AN OPTION. ALSO THE "IMAGE" IS REPLACED WITH THE REMAINDER OF THE COMMAND STRING. SO A COMMAND THAT HAS TWO OPTIONS WILL HAVE 3 CALLS TO "PARSER". FINALLY THE ROUTINE IS A FUNCTION TO INDICATE END OF PROCESSING, I.E. EITHER THE SEMI-COLON WAS ENCOUNTERED OR COLUMN 80. THE RETURN IS FALSE FOR THE END OF PROCESSING SIGNAL.

## RECORD POSITIONING

THE TWO METHODS OF RECORD POSITIONING ARE RELATIVE AND ABSOLUTE. BOTH METHODS MUST DEAL WITH THE STRUCTURE OF THE DATA SET UNDER UMTSS.

ABSOLUTE POSITIONING REQUIRES COUNTING THE RECORDS IN THE DATA SET STARTING FROM THE BEGINNING. SINCE NEITHER THE UMTSS TABLES NOR THE LIBRARY STRUCTURE HAS THE CURRENT RECORD NUMBER, THE PRIMARY LT MUST BE ACCESSED TO SEQUENTIALLY COUNT THE NUMBER OF RECORDS IN THE DATA SET. THIS INVOLVES ACCESSING EACH SECONDARY LT FOR THE INFORMATION. EACH LT HAS THE NUMBER OF RECORDS IN IT BUT THERE IS NO FIELD IN THE PLT'S ASSOCIATED LT BLOCK FOR THIS NUMBER (IT WOULD DOUBLE THE NUMBER OF ACCESSES TO THE DISK IN ORDER TO KEEP THE FILES UPDATED).

THE RELATIVE METHOD USES THE RECORD KEY TO FIND THE POSITION OF THE RECORD. THE KEY IS SEARCHED THROUGH THE ASSOCIATED LT TABLES TO FIND THE PROPER SECONDARY LT. THEN THAT TRACK IS READ IN AND THE KEY IS AGAIN USED TO FIND ITS POSITION WITHIN THE TRACK.

COMPARING BOTH METHODS, THE RELATIVE METHOD IS SUPERIOR TO THE ABSOLUTE METHOD. THE FORMER METHOD REQUIRES ONLY TWO ACCESSES WHILE THE LATTER METHOD REQUIRES AT LEAST TWO AND POSSIBLY MANY MORE IF THE RECORD IS NEAR THE END OF A LARGE DATA SET. THE TRADE-OFFS ARE SIMILAR TO SEQUENTIAL PROCESSING VERSUS RANDOM PROCESSING.

## BLOCK MANAGEMENT

THE MOST IMPORTANT ROUTINES TO THE LIBRARY SYSTEM ARE THE BLOCK MANAGEMENT PROGRAMS. THESE PROCEDURES PROVIDE THE NECESSARY TABLE LOOK-UP PROCEDURE AND THE VITAL BLOCK UPDATING FUNCTIONS.

THE "MAP" ROUTINE PROVIDES A STRUCTURE MAPPING OF A LOGICAL TRACK. FROM THE BLOCK INFORMATION WITHIN THE TRACK, THE ROUTINE CALCULATES AND SAVES PERTINENT INFORMATION REGARDING THE TRACK. THESE VALUES ARE CONTAINED THE GLOBALS "ABLK", THE RELATIVE ADDRESS OF EACH BLOCK WITHIN THE ARRAY (EITHER "BUFF" OR "TABLE"), "ASUBS", THE RELATIVE ADDRESS OF THE SUB-BLOCKS, "#BLKS", THE NUMBER OF WORDS IN THE BLOCK, "#DATA", NUMBER OF WORDS IN DATA AREA OF THE SUB-BLOCK, "#KEYS", NUMBER OF WORDS IN THE KEY AREA, "#LOCK", NUMBER OF WORDS IN THE LOCKOUT AREA, "#SUBS", NUMBER OF SUB-BLOCKS IN THE BLOCK, AND "#TEXT", THE NUMBER OF WORDS IN THE TEXT AREA.

"BDAM" PROVIDES THE INSERTION AND DELETION OF SUB-BLOCKS IN BLOCKS 3 AND 4. IT ALSO PROVIDES RETRIEVAL OF SUB-BLOCKS FROM THESE BLOCKS.

FINALLY, "ISAM" SEARCHES THE DIRECTORY TABLES GIVEN A KEY AND FINDS THE POSITION OF THE KEY WITHIN THE TABLE. THE SEARCH IS STRICTLY SEQUENTIAL. IT RETURNS A FLAG TO INDICATE THE KIND OF POSTION, I.E. LESS THAN A KEY, EQUAL TO A KEY, BETWEEN TWO KEYS, OR OUTSIDE THE RANGE.

## UHISS COMMUNICATIONS

THE LIBRARY PROGRAMS COMMUNICATE WITH THE TERMINAL OR THE MONITOR VIA POSTS AND WAITS. THE "SYSTEM" ROUTINE INFORMS THE MONITOR THAT THE LIBRARY PROGRAM HAS CRASHED, THE "CONSOLE" PROGRAM COMMUNICATES WITH THE TERMINAL, THE "POST" ROUTINE IS THE BASIC TRANSMITTER TO THE MONITOR, AND THE "WAIT" PROGRAM TELLS THE MONITOR THAT THE LIBRARY PROGRAMS IS FINISHED.

THE SURF BUFFER IS STORED IN A WORK AREA. ALL MANIPULATION WITH THE BUFFER IS DONE WITH RESPECT TO THE WORK AREA. WHEN THE RESULTS ARE TO BE PASSED BACK VIA THE SURF BUFFER, THE LIBRARY MUST WAIT TILL THE BUFFER IS FREE. THEN THE BUFFER IS STUFFED FROM THE WORK AREA. THIS OCCURS IN 2 ROUTINES: CONSOLE AND LINK.

## OS COMMUNICATIONS

IN ORDER TO MOVE DATA FROM ONE PROTECT REGION INTO ANOTHER, UHTSS HAS AN SVC WHICH WILL PUT THE PROGRAM IN ZERO PROTECT KEY. THE ROUTINES INVOLVED ARE:

GET_INTO_PROTECT_KEY_ZERO.

GET_OUT_OF_PROTECT_KEY_ZERO.

## IDIOSYNCRACIES

MOST OF THE IDOSYNCRACIES ARE NOT MAJOR ROADBLOCKS BUT AT TIMES CONSTITUTE AN UNNECESSARY DIGRESSION IN PROGRAMMING TO SOLVE THEM. MOST OF THE SOLUTIONS HAVE BEEN DONE USING "INLINE" CODING.

### COREWORD

THE XPL IMPLEMENTATION OF COREWORD REQUIRES A WORD ADDRESS WHILE NOWHERE WITHIN THE STRUCTURE OF XPL IS THERE ANY FACILITY FOR OBTAINNING A WORD ADDRESS DIRECTLY. AS A RESULT OF THIS, "MAPPER" PROVIDES BYTE ADDRESSING FOR WORD ARRAYS.

### STRINGS

STRINGS ARE PASSED AS STRING DESCRIPTORS WITH AN 8 BIT LENGTH AND A 24 BIT ADDRESS FIELD. THERE ARE ENTRIES WITHIN THE UHTSS TABLE THAT SHOULD BE TREATED AS STRINGS BUT BECAUSE THERE ARE NO EQUIVALENCE OR DEFINED STATEMENTS, WE MUST PERFORM THE MAPPING OURSELVES. THE ROUTINE "STRING" HAS A PSUEDO-DESCRIPTOR AS AN ARGUMENT AND RETURNS THIS DESCRIPTOR AS THE PROPER VALUE FOR A CHARACTER PROCEDURE. THUS USING "STRING" AS A FUNCTION WITH A WORD THAT HAS THE HIGH-ORDER BYTE SET TO SOME LENGTH AND THE LOW-ORDER 3 BYTES TO SOME ADDRESS, WE CAN USE ELEMENTS OF AN ARRAY AS AN XPL CHARACTER STRING.

SINCE DATA FIELDS MUST BE PHYSICALLY MOVED, THE XPL CONVENTIONS OF SETTING DESCRIPTORS EQUAL IS UNSATISFACTORY. "MOVEBYTES" MOVES BLOCKS OF UP TO 256 BYTES FROM "SOURCE" TO "TARGET". BOTH SOURCE OR TARGET ARE CORE ADDRESSES WHICH ARE USUALLY STRING DESCRIPTORS. "MOVER" CALLS "MOVEBYTES" BUT MOVES IN TERMS OF WORDS.

THERE IS PERHAPS A BUG WITHIN XPL. THERE SEEMS TO BE SOME SORT OF FAILURE TO CONCATENATE PROPERLY WHEN THE FREE STRING AREA IS OVERFLOWED AND COMPACTIFICATION MUST OCCUR. FOR THIS REASON THERE WILL BE NO CONCATENATION WITHIN THE PROGRAMS.

### EXTERNAL

SINCE XPL LACKS THE ABILITY TO BRANCH TO ROUTINES WHOSE ADDRESSES WERE PASSED AS PARAMETERS, "MASS" WAS DESIGNED TO PERFORM A SIMPLE BALR.

## SUBROUTINE ANATOMY

THIS SECTION DESCRIBES EACH SUBROUTINE USED IN THE LIBRARY PROGRAM. EACH DESCRIPTION INCLUDES A NARRATIVE ON THE FUNCTIONS AND QUIRKS, A LIST OF PROCEDURES IT INVOKES AND FINALLY A LIST OF GLOBALS IT REFERENCES. THE ORDER OF THE SUBROUTINES IS THE ORDER REQURIED BY XPL AND MY OWN PERSONAL QUIRK.

### SIRING

DESCRIPTIVE NOTES:
TAKES A PSEUDO-DESCRIPTOR AND RETURNS IT AS AN XPL DESCRIPTOR. A PSEUDO-DESCRIPTOR IS A WORD WITH THE HIGH ORDER BYTE BEING THE LENGTH OF THE STRING MINUS 1 (NULL STRINGS ARE REPRESENTED BY A ZERO WORD). THE LOW ORDER 3 BYTES IS THE MACHINE ADDRESS.

### MAPPER

DESCRIPTIVE NOTES:
PERFORMS THE SAME FUNCTION AS COREWORD. DIFFERENCE IS THAT IT USES AN ADDRESS THAT IS A PARAMETER TO A SUBROUTINE.

INVOKED PROCEDURES:
INLINE

### ZAP

DESCRIPTIVE NOTES:
BLANKS OUT A STRING. THE DESCRIPTOR MUST POINT TO AN AREA BELOW FREEBASE.

INVOKED PROCEDURES:
INLINE

### MOVEBYTES

DESCRIPTIVE NOTES:
MOVES UP TO 256 BYTES AROUND. THE REASON FOR THE BCTR 3,0 IS THAT LIMITED IS THE NUMBER OF BYTES TO BE MOVED, BUT THE MACHINE REQUIRES THAT THE LENGTH BE THE LENGTH OF THE STRING MINUS ONE.

INVOKED PROCEDURES:
INLINE

### MOVEB

DESCRIPTIVE NOTES:
MOVES BLOCKS OF WORDS ABOUT. TRIES TO OPTIMIZE BY

MOVING BLOCKS OF 256 BYTES AT A TIME.

INVOKED PROCEDURES:
MOVEBYTES

LEADING

DESCRIPTIVE NOTES:
PUTS LEADING BLANKS INTO A STRING. THE TARGET STRING MUST BE BELOW FREEBASE.

INVOKED PROCEDURES:
LENGTH, MOVEBYTES, STRING, ZAP

TRAILING

DESCRIPTIVE NOTES:
PUTS TRAILING BLANKS INTO A STRING. THE TARGET STRING MUST BE BELOW FREEBASE.

INVOKED PROCEDURES:
LENGTH, MOVEBYTES, STRING, ZAP

FFX

DESCRIPTIVE NOTES:
CHARACTER FUNCTION

PERFORMS BINARY TO HEXADECIMAL CHARACTER STRING CONVERSION FOR OUTPUTTING.

INVOKED PROCEDURES:
SUBSTR

BACKMOVE

DESCRIPTIVE NOTES:
MAKES ROOM FOR A BLOCK BY MOVING THE BLOCKS AFTER IT A BLOCK UP.

INVOKED PROCEDURES:
MOVE

LOGGING

DESCRIPTIVE NOTES:
LOG THE TIME, TERMINAL, DATA SET NAME, AND MESSAGE, ONLY IF SYSTEM TRACE IS LESS THAN 1.

INVOKED PROCEDURES:
STRING, SUBSTR

REFERENCED GLOBALS:
SYSTEM TRACE

WIO

DESCRIPTIVE NOTES:
WRITE TO THE OS SYSTEM TYPEWRITER/OPERATOR, INLINES THE SVC. MAKE SURE TO SAVE GENERAL REGISTER 15 ELSE XPL WILL BE SCREWED UP.

INVOKED PROCEDURES:
INLINE

GET_INTO_PROTECT_KEY_ZERO

DESCRIPTIVE NOTES:
PUTS THE PROGRAM IN PROTECT KEY ZERO BY ISSUING A SPECIAL SVC.

INVOKED PROCEDURES:
INLINE

GET_OUT_OF_PROTECT_KEY_ZERO

DESCRIPTIVE NOTES:
GETS THE PROGRAM OUT OF PROTECT KEY ZERO, ISSUES A SPECIAL SVC.

INVOKED PROCEDURES:
INLINE

POST

DESCRIPTIVE NOTES:
SIGNAL THE COMPLETION OF AN EVENT.

INVOKED PROCEDURES:
INLINE

WAIT

DESCRIPTIVE NOTES:
AWAIT THE COMPLETION OF THE EVENT.

INVOKED PROCEDURES:
INLINE

WAIT_FOR_BUFFER

DESCRIPTIVE NOTES:
TELL CMON THAT I'M GOING TO USE THE SURF BUFFER. WAIT TILL IT'S OK.

INVOKED PROCEDURES:
POST, WAIT

REFERENCED GLOBALS:
CMON_QECB, IOSTAT, ORIGIN, TERMAD

REQUEST_INPUT

DESCRIPTIVE NOTES:
ASK TMON TO GIVE ME INPUTS WITHOUT EXAMINING THE CONTENTS OF THE BUFFER. WE ARE NO LONGER IN COMMAND MODE BUT IN MODE 52 (AT THE MOMENT INSERT MODE).

INVOKED PROCEDURES:
POST

REFERENCED GLOBALS:
COMPLETED STATUS TERMAD

CONSOLE

DESCRIPTIVE NOTES:
WRITE TO THE TERMINAL. NOTE THAT WE MUST WAIT UNTIL THE SURF BUFFER IS FREE FOR USAGE. IF THE SUBROUTINE NAME 'X' HAS AN '*' AS THE FIRST CHARACTER, THEN THE MESSAGE GOES TO THE ACTIVE_LINE (OF THE 2260'S). OTHERWISE IT GOES TO THE FORMATTING LINE. NOTE THAT THE SURF BUFR IS STUFFED WITH THE CONTENTS OF THE WORKING BUFFER.

INVOKED PROCEDURES:
BYTE, LENGTH, LOGGING, MOVE, MOVEBYTES, POST, TRAILING, WAIT, WAIT_FOR_BUFFER, ZAP

REFERENCED GLOBALS:
APAGE, BUFFER, BUFR, TERMAD

SYSTEMX

DESCRIPTIVE NOTES:
KILL MY PART OF THE PROGRAM. WE HAVE A DISASTROUS ERROR. FORCE LOGGING.

INVOKED PROCEDURES:
OPERATOR, WAIT

REFERENCED GLOBALS:
SYSTEM_TRACE

LIBRARY_CRASH

DESCRIPTIVE NOTES:
THE COMMAND WAS NOT IN THE BUFFER. KILL MY PROGRAM.

INVOKED PROCEDURES:
SYSTEM

CONVERT_BINARY

DESCRIPTIVE NOTES:
LOGICAL FUNCTION

CONVERT A CHARACTER STRING TO A BINARY NUMBER, INCLUDING NEGATIVE NUMBERS. IF THE RETURN IS TRUE, THEN THERE IS A CONVERSION ERROR. OTHERWISE THE ANSWER IS RETURNED IN 'VALUE'.

INVOKED PROCEDURES:
BYTE, LENGTH, OPERATOR

REFERENCED GLOBALS:
VALUE

INDEX2

DESCRIPTIVE NOTES:
FIXED FUNCTION

PERFORMS A SEARCH OF A STRING FOR A GIVEN SUBSTRING WITH PARAMETERS TO TELL WHERE TO START AND STOP SEARCHING AND HOW TO INCREMENT THE SEARCH.

INVOKED PROCEDURES:
LENGTH, SUBSTR

INDEX1

DESCRIPTIVE NOTES:
FIXED FUNCTION

SAME AS "INDEX" FUNCTION OF PL/I. CAN'T USE "INDEX" BECAUSE IT'S A LITERAL.

INVOKED PROCEDURES:
INDEX2, LENGTH

PARSING

DESCRIPTIVE NOTES:
LOGICAL FUNCTION

TAKES A STRING AND RETURNS THE KEYWORD/OPTION USING THE SYNTAX DEFINED FOR THE COMMANDS. HANDLES APOSTROPHES OK. THE PARAMETER DELIMITER IS A BLANK OR COMMA. THE COMMAND DELIMITER IS A SEMI-COLON. CONTINUING MAY NOT BE RESET PROPERLY. THAT IS WHY IN TSS_LIBRARY ON THE FIRST CALL, IT IS RESET.

*WARNING* PARSING DESTROYS IMAGE IF LITERAL STRINGS ARE USED. THE TECHNIQUE USING PARSING MAKES IMAGE = BUFFER. NOTE THAT BUFFER IS AN ARRAY.

INVOKED PROCEDURES:
BYTE, INDEX1, LENGTH OPERATOR, SUBSTR

REFERENCED GLOBALS:
IMAGE, KEYWORD, OPTION, VALUE, CONTINUING

MAP

DESCRIPTIVE NOTES:
ANALYZES EITHER BUFFERS AND FILL THE MAPPING ARRAYS WITH THE APPROPRIATE VALUES.

INVOKED PROCEDURES:
MAPPER, SYSTEM

REFERENCED GLOBALS:
#BLKS, ABLK, ASUBS, #SUBS, #DATA, #KEYS, #LOCK, #TEXT, CURRENT_BUFFER, FREE_AREA

BDAM

DESCRIPTIVE NOTES:
MANIPULATES SUBBLOCKS OF BLOCKS 3 OR 4. INSERTS, DELETES, AND RETRIEVES. IF THE SUBBLOCK# IS OUT OF RANGE, THE SUBBLOCK# IS RESET TO THE BOUNDARY LIMIT.

INVOKED PROCEDURES:
BACKMOVE, COREWORD, MAP, MOVE, OPERATOR

REFERENCED GLOBALS:
#TEXT, ASUBS, CURRENT_BUFFER, FREE_AREA, #BLKS, #SUBS

ISAM

DESCRIPTIVE NOTES:
FIXED FUNCTION

SEARCHES THROUGH A LIST OF KEYS TO SEE IF A GIVEN KEY IS WITHIN A RANGE OF THEM. RETURNS FLAG AND SUB#.

INVOKED PROCEDURES:
STRING, OPERATOR

REFERENCED GLOBALS:
#KEYS, #TEXT, ASUBS, #DATA, FLAG, SUB#, CONDCODE

FIND_BIT

DESCRIPTIVE NOTES:
FIXED FUNCTION

TRIES TO FIND A FREE BIT GIVEN A FREE CYLINDER BLOCK. CALCULATES THE APPROPRIATE LOGICAL TRACK NUMBER. RETURNS ZERO OTHERWISE.

INVOKED PROCEDURES:
FREE_BLK

GETBB

DESCRIPTIVE NOTES:
FIXED FUNCTION

SEARCHES THROUGH THE CORE TABLE'S BLOCK 4 FROM THE CENTRAL CYLINDER. RETURNS THE LT#.

INVOKED PROCEDURES:
BDAM, FIND_BIT. MAP, SYSTEM

REFERENCED GLOBALS:
ATABLE, #SUBS ABUFF, REWRITE

FREEIBB

DESCRIPTIVE NOTES:
FREES A LOGICAL TRACK BY RESETTING THE APPROPRIATE BIT IN THE PROPER FREE CYLINDER BLOCK.

INVOKED PROCEDURES:
BDAM, MAP

REFERENCED GLOBALS:
ATABLE, LT#, ABUFF, REWRITE

IBACK_CHECK

DESCRIPTIVE NOTES:
MAKE SURE THAT THE LT# IS WITHIN THE BOUNDS OF THE DISK FILE.

INVOKED PROCEDURES:
SYSTEM

REFERENCED GLOBALS:
LAST_TRACK#, LT#

BITEY

DESCRIPTIVE NOTES:
DETERMINES IF THE DTOC SHOULD BE WRITTEN. IF SO, WRITES IT.

INVOKED PROCEDURES:
FILE, TRACK_CHECK

BEADY

DESCRIPTIVE NOTES:
READS IN THE DESIRED LOGICAL TRACK. CHECKS FIRST TO SEE IF IT IS ALREADY IN CORE. IF IT IS JUST ACT AS IF IT WAS READ IN.

INVOKED PROCEDURES:

FILE, MAP, RITEY

REFERENCED GLOBALS:
LT#, IOCOUNT, BUFF, LAST_LT#

READX

DESCRIPTIVE NOTES:
READS IN THE TRACK AND TRIES TO GET THE HIGH AND LOW KEYS (CANNOT BE DONE FOR LOADABLE DATA SETS).

INVOKED PROCEDURES:
BDAM, MOVE, READY

REFERENCED GLOBALS:
#TEXT, LO, HI

RITEX

DESCRIPTIVE NOTES:
WRITES THE LOGICAL TRACK ONTO DISK.

INVOKED PROCEDURES:
FILE, RITEY, TRACK_CHECK

REFERENCED GLOBALS:
LT#, ABLK, BUFF, IOCOUNT, LAST_LT#

ALLOCATE_DSNODE

DESCRIPTIVE NOTES:
OBTAINS A FREE DSNODE FOR A TERMINAL.

INVOKED PROCEDURES:
SYSTEM

REFERENCED GLOBALS:
DSNODE, DSPNTR, DSHEAD, DC81, DC82, FREEHEAD, COUNT

FREE_DSNODE

DESCRIPTIVE NOTES:
FREES A DSNODE LIST AND PUTS EACH DSNODE BACK ON THE FREE DSNODE LIST, EXCEPT THE LAST ONE. IT IS ALWAYS ALLOCATED TO THE SURF TERMINAL.

REFERENCED GLOBALS:
DSNODE, DSHEAD, DSPNTR, FREEHEAD, COUNT

INITIALIZE DSNODE

DESCRIPTIVE NOTES:
INITIALIZES DATA SET PARAMETER POINTERS.

REFERENCED GLOBALS:

DSNODE, FREEHEAD, DSPNTR, DSHEAD, LO, CURRENT, HI, STOPKEY, CURRENT_DSNODE

INITIALIZE

DESCRIPTIVE NOTES:
READ IN THE CURRENT DTOC, INITIALIZE POINTERS, GET THE DATE, SET UP THE FREE DSNODE LIST AND ALLOCATE ONE TO EACH TERMINAL. CALCULATE TODAY'S DATE (TRY TO HANDLE LEAP YEAR, MIGHT NOT WORK).

INVOKED PROCEDURES:
ALLOCATE_DSNODE, DATE, FILE, GET_FILE, MAP, MOVE, STRING, SUBSTR, SYSTEM

REFERENCED GLOBALS:
ATABLE, APADDING, USE_COUNT, LAST_TRACK#, #SUBS, ABUFF, BUFFER, AFREE, APAGE, DATE_EXP, DATE_NOW, NEXT-KEY, DATE_LIMIT, DATE_TODAY, FREE_LOGICAL_TRACK, LT#, FREE_AREA, DSNODE, DSHEAD, DSPNTR, NUM_TERM, ENTRY_LENGTH

REINITIALIZE

DESCRIPTIVE NOTES:
FREES THE LT AND WRITES A FREE LOGICAL TRACK OUT.

INVOKED PROCEDURES:
FREETRK, MOVE, WRITE_FILE

REFERENCED GLOBALS:
ABUFF, A_FREE_IMAGE

LOOK_AT_TABLE

DESCRIPTIVE NOTES:
FIXED FUNCTION

EXAMINES THE CORE TABLE FOR THE JOB/ACCOUNT NUMBER.

INVOKED PROCEDURES:
ISAM, MAP, MOVE, STRING

REFERENCED GLOBALS:
AJOB, APAGE, ATABLE, FLAG

LOOK_AT_PAGE

DESCRIPTIVE NOTES:
FIXED FUNCTION

EXAMINES THE DIRECTORY PAGE FOR THE JOB/ACCOUNT NUMBER AND DSNAME.

INVOKED PROCEDURES:
BDAM, GET_FILE, ISAM, MOVE, STRING

REFERENCED GLOBALS:
SUB#, LT#, APAGE, AJOB, ADSET, FLAG, ABUFF

CREATE_LABEL_BLOCK

DESCRIPTIVE NOTES:
FILLS IN THE LABEL BLOCK WITH THE RIGHT STUFF.

INVOKED PROCEDURES:
MOVE

REFERENCED GLOBALS:
ABLK, BUFF, AJOB, ADSET, AUSER_NAME, LT#

CREATE_DATA_SET_BLOCK

DESCRIPTIVE NOTES:
FILLS IN THE DATA SET BLOCK FOR THE DIRECTORY PAGE.

INVOKED PROCEDURES:
MOVE

REFERENCED GLOBALS:
LT#, APAGE, DATE_TODAY, DATE_LIMIT, AUSER_NAME,
AJOB, ADSET

DATA_SET_ERROR

DESCRIPTIVE NOTES:
LOGICAL FUNCTION

CHECK TO SEE IF A DATA SET HAS ALREADY BEEN
SELECTED.

INVOKED PROCEDURES:
OPERATOR, STRING, SYSTEM

REFERENCED GLOBALS:
ADSET

BAD_BLOCK_SIZE

DESCRIPTIVE NOTES:
LOGICAL FUNCTION

CHECK TO SEE IF THE TEXT SIZE IS LESS THAN OR EQUAL
TO 20 WORDS, SINCE INSERT AND DELETE USES SCRATCH
ARRAYS OF ONLY 20 WORDS.

INVOKED PROCEDURES:
OPERATOR

REFERENCED GLOBALS:
LT#, #TEXT

CHECK_KEY

DESCRIPTIVE NOTES:
LOGICAL FUNCTION

CHECK FOR KEYS OF LENGTH >6. FORCE BAD CONDCODE IF IT IS.

INVOKED PROCEDURES:
LENGTH, OPERATOR LLT REFERENCED GLOBALS:
CONDCODE

INPUTER

DESCRIPTIVE NOTES:
LOGICAL FUNCTION

CHECK TO SEE IF THE COMMAND HAS THE DATA SET NAME. IF THE COMMAND COMES FROM ANYWHERE ELSE, ALLOCATE A NEW DSNODE. PRIMARILY FOR SELECT AND CREATE.

INVOKED PROCEDURES:
ALLOCATE_DSNODE, INITIALIZE_DSNODE, LIBRARY CRASH, OPERATOR, PARSING, TRAILING

REFERENCED GLOBALS:
ORIGIN, IMAGE, BUFFER, OPTION, ADSET

RELATIVE_POSITION

DESCRIPTIVE NOTES:
LOGICAL FUNCTION

TRIES TO POSITION THE FILE AT THE GIVEN KEY. SETS FLAG AND SUB# PROPERLY. FIRST SEE IF IN CORE, IF NOT READING THE PLT#. THEN GET THE APPROPRIATE SECONDARY LOGICAL TRACK.

INVOKED PROCEDURES:
BDAM, GET_FILE, ISAM, READ_FILE, SYSTEM

REFERENCED GLOBALS:
LT#, DCB1, DCB2, CURRENT, LO, HI, ABUFF, APAGE

ABSOLUTE_POSITION

DESCRIPTIVE NOTES:
LOGICAL FUNCTION

GIVEN THE RECORD NUMBER, POSITIONS THE FILE THERE. READ IN SEQUENCE EACH LT# AND COUNT HOW MANY CARDS

THERE ARE UNTIL WE REACH THE APPROPRIATE NUMBER.

INVOKED PROCEDURES:
BDAM, MOVE, READ_FILE

REFERENCED GLOBALS:
LT#, DCB1, DCB2, #TEXT, SUBS, ASUBS, APADDING, SUB#, RECORD#, APAGE, CURRENT

GET_RECORD

DESCRIPTIVE NOTES:
LOGICAL FUNCTION

GET THE RECORD CORRESPONDING TO THE NEXT KEY. SHOULD MOVE THE CURRENT KEY INTO THE NEXT KEY AND AT THE END MOVE THE NEXT KEY INTO THE CURRENT KEY.

INVOKED PROCEDURES:
BDAM, GET_FILE, ISAM, MOVE, READ_FILE, RELATIVE_POSITION

REFERENCED GLOBALS:
NEXT_KEY, CURRENT, MAXKEY, ABUFFER, #SUBS, LT#, DCB1, DCB2, ABUFF, SUB#, APAGE

LIBRARY_SELECT

DESCRIPTIVE NOTES:
SINGLE SERVICE REQUEST

AFTER INVESTIGATING THE DTOC AND PAGE, UPDATE THE DATE SELECTED FIELD IN THE PAGE. CHECK IF HIS DATA SET IS STILL LEGITIMATE. POSITION THE FILE AT THE FIRST RECORD IF IT'S A CARD IMAGE DATA SET.

INVOKED PROCEDURES:
BDAM, GET_FILE, INPUTER, LOOK_AT_PAGE, LOOK_AT_TABLE, MOVE, OPERATOR, WRITE_FILE

REFERENCED GLOBALS:
LT#, DCB1, DCB2, DATE_EXP, OPTION, SUB#, APAGE, DATE_NOW, DATE_TODAY, #TEXT, ADSET, NO_DATA_SET, COUNT, CONDCODE

LIBRARY_CREATE

DESCRIPTIVE NOTES:
SINGLE SERVICE REQUEST

CREATES A DATA SET BLOCK AND UPDATES THE APPROPRIATE PAGE. CHECK IF PAGE OVERFLOW OCCURS (NO DYNAMIC MECHANISM YET). WRITES A FREE LOGICAL TRACK IMAGE WITH THE PROPER FORMAT.

INVOKED PROCEDURES:
BDAM, CREATE_DATA_SET_BLOCK, CREATE_LABEL_BLOCK, GETTRK, INPUTER, LOOK_AT_PAGE, LOOK_AT_TABLE, MAP, MOVE, OPERATOR, WRITE_FILE

REFERENCED GLOBALS:
NO_DATA_SET, ADSET, CONDCODE, LT#, DCB1, DCB2, COUNT, HI, CURRENT, PAGE, A_FREE_IMAGE, ABUFF, #BLKS, #TEXT

LIBRARY_SCRATCH

DESCRIPTIVE NOTES:
MULTI-SERVICE REQUEST.

SCRATCH ALL THE LOGICAL TRACKS ASSOCIATED WITH THE DATA SET. THE PAGE DATA SET BLOCK IS WIPED OUT FIRST. THE PLT IS DONE THE LAST.

INVOKED PROCEDURES:
BDAM, GET_FILE, INPUTER, LOOK_AT_PAGE, LOOK_AT_TABLE, MOVE, OPERATOR, REINITIALIZE, WRITE_FILE

REFERENCED GLOBALS:
COUNT, DCB1, DCB2, PAGE, LT#, #SUBS, APADDING, ADSET, NO_DATA_SET, CONDCODE

MASS

DESCRIPTIVE NOTES:
BRANCHES TO ANY PROCEDURE.

INVOKED PROCEDURES:
INLINE

OUTPUTER

DESCRIPTIVE NOTES:
MULTI-SERVICE REQUEST. BREAKS UP COMMANDS INTO ONES WITH KEY WORDS OR ONES WITH POSITIONAL PARAMETERS. POSITIONS THE FILE TO APPROPRIATE RECORD THEN OUTPUTS THE BUFFER TO THE APPROPRIATE ROUTINES.

INVOKED PROCEDURES:
ABSOLUTE_POSITION, BAD_BLOCK_SIZE, BDAM, CONVERT_BINARY, DATA_SET_ERROR, GET_FILE, GET_RECORD, INDEX1, ISAM, LEADING, LIBRARY_CRASH, LOOK_AT_TABLE, MASS, MOVE, MOVEBYTES, OPERATOR, PARSING, READ_FILE, STRING, SUBSTR, SYSTEM, TRAILING, ZAP

REFERENCED GLOBALS:
COUNT, BUFFER, APAGE, AUSER_NAME, IMAGE, SUBIOTYP, #SUBS, DCB1, DCB2, LT#, AJOB, IOCOUNT, TIMER,

ADSET, ACURRENT

LIBRARY HASP

DESCRIPTIVE NOTES:
OUTPUTS TO THE INTERNAL READER OF HASP. BLANK OUT
73-80. IT WILL SCREW UP BASIC PROGRAMS BUT HECK IT

INVOKED PROCEDURES:
OPERATOR, ZAP, STRING

REFERENCED GLOBALS:
CONDCODE, HASP_USER, BUFFER, ABUFFER72

LIBRARY PRINT

DESCRIPTIVE NOTES:
OUTPUTS TO THE INTERNAL READER OF HASP.

INVOKED PROCEDURES:
STRING

REFERENCED GLOBALS:
CONDCODE, PRINT_USER, BUFFER

LIBRARY PUNCH

DESCRIPTIVE NOTES:
OUTPUTS TO THE INTERNAL READER OF HASP.

INVOKED PROCEDURES:
STRING

REFERENCED GLOBALS:
CONDCODE, PUNCH_USER, BUFFER

LIBRARY DISPLAY

DESCRIPTIVE NOTES:
OUTPUTS TO THE TERMINAL.

INVOKED PROCEDURES:
MOVE, OPERATOR

REFERENCED GLOBALS:
APAGE, BUFFER

LIBRARY INSERT

DESCRIPTIVE NOTES:
INSERTS A CARD IMAGE INTO A USER'S DATA SET.
HANDLES TRACK OVERFLOW BY WRITING THE LAST 15 CARDS
ONTO A NEW TRACK.

INVOKED PROCEDURES:

BAD_BLOCK_SIZE, BDAM, CREATE_LABEL_BLOCK,
DATA_SET_ERROR, GET_FILE, GETTRK, ISAM, MAP, MOVE,
OPERATOR, READ_FILE, RELATIVE_POSITION, WRITE_FILE

REFERENCED GLOBALS:
CURRENT, LO, #BLKS, #TEXT, #SUBS, APAGE, ABUFFER,
APADDING, LT#, DCB1, HI, ABUFF, A_FREE_IMAGE,
BAD_BLOCK_SIZE, BDAM, BYTE, COREBYTE

FREE_SCAN_INSERT

DESCRIPTIVE NOTES:
ALLOWS FREE FORM FOR INSERTION. THE SCAN LOOKS FOR THE FIRST NON-BLANK SEQUENCE OF CHARACTERS WITHIN COLUMNS 1 TO 8. THE FIRST BLANK AFTER THE NON-BLANK STRING TERMINATES THAT STRING. IT BECOMES THE KEY. THE NEXT 72 CHARACTERS AFTER THAT BLANK BECOMES THE TEXT.

INVOKED PROCEDURES:
COREBYTE, LEADING, MOVE, MOVEBYTES, OPERATOR,
STRING

REFERENCED GLOBALS:
ABUFFER, CONDCODE, APAGE, CURRENT

PRE_LIBRARY_INSERT

DESCRIPTIVE NOTES:
DETERMINES IF STANDARD COMMAND MODE OR KEYLESS MODE. IF IT IS KEYLESS MODE THEN DECIDE TO EITHER AUTOMATICALLY GENERATE THE KEYS OR USE THE FREE_SCAN_INSERT TECHNIQUE (OPTION 'BASIC'). THE SENTINEL STRING TERMINATES THE KEYLESS MODE.

INVOKED PROCEDURES:
BAD_BLOCK_SIZE, BYTE, CONVERT_BINARY, COREBYTE,
DATA_SET_ERROR, FREE_SCAN_INSERT, LEADING,
LIBRARY_CRASH, LIBRARY_INSERT, MOVE, OPERATOR,
PARSING,STRING, SUBSTR, TRAILING

REFERENCED GLOBALS:
SUBIOTYP, BUFFER, COUNT, COUNT2, STOPKEY, CURRENT

DELETE_LOOP

DESCRIPTIVE NOTES:
DELETES CARDS SUBX TO SUBY. IF THE LAST CARD DELETED WAS THE HI KEY THEN WE MUST FIX THE DSNODE, THE BLOCK 3 POINTER AND THE DIRECTORY POINTERS. IT IS AN INTERNAL PROCEDURE USING INTERNAL GLOBALS (SUBX, SUBY, AND SUB1).

INVOKED PROCEDURES:
BDAM, GET_FILE, ISAM, MOVEBYTES, STRING, WRITE_FILE

REFERENCED GLOBALS:
BUFFER, HI, LT#, DCB1, PAGE

IBACK_UNDERFLOW

DESCRIPTIVE NOTES:
EITHER THERE ARE NO CARDS IN THE TRACK OR THERE IS ONLY ONE LEFT. FOR CASE 1, THROW THE TRACK AWAY AND REMOVE IT FROM THE DIRECTORY. FOR CASE 2, SAVE THE LAST CARD, THROW THE TRACK AWAY, AND PUT THE LAST CARD BACK IN (INSERT WILL AFFECT TRACKS AFTER THE ONE WE HAVE THROWN AWAY).

LIBRARY_DELETE

DESCRIPTIVE NOTES:
MULTI-SERVICE REQUEST.

DELETES A CARD IMAGE FROM THE USER'S DATA SET. THERE ARE 3 POSSIBLE CASES. THEY ARE THAT THE START AND STOP KEYS ARE: ON THE SAME TRACK, 2) ON ADJACENT TRACKS, OR 3) SEPARATED BY 1 OR MORE TRACKS. THE ALGORITHM IS TO TAKE CASE 3, REDUCE IT TO CASE 2. TAKE CASE 2 AND REDUCE IT TO CASE 1. IN THIS MANNER THE NUMBER OF DISK READ/WRITES IS MINIMIZED. THE FIRST IMPLEMENTATION DELETED A CARD AT A TIME. THIS IS CLEARLY SIMPLE BUT GROSSLY INEFFICIENT.

INVOKED PROCEDURES:
BAD_BLOCK_SIZE, BDAM, DATA_SET_ERROR, DELETE_LOOP, GET_FILE, ISAM, LEADING, LIBRARY_CRASH, LIBRARY_INSERT, MOVE, OPERATOR, PARSING, REINITIALIZE, STRING, TRACK_UNDERFLOW, WRITE_FILE

REFERENCED GLOBALS:
LT#, DCB1, DCB2, COUNT, APAGE, BUFFER, IMAGE, SUBIOTYP

LIBRARY_LINK

DESCRIPTIVE NOTES:
SELECTS AN XPL LOADABLE DATA SET AND AUGMENTS THE CONTROL RECORD WITH THE NUMBER OF LOGICAL TRACKS IN THE DATA SET. THE CONTROL RECORD IS PASSED IN THE BUFFER.

INVOKED PROCEDURES:
BDAM, GET_FILE, LIBRARY_SELECT, MOVE, POST, WAIT

REFERENCED GLOBALS:
CONDCODE, DCB1, DCB2, LT#, #SUBS, ASUBS, PAGE.BUFR

LIBRARY_LOAD

DESCRIPTIVE NOTES:
READS IN A LOGICAL TRACK AND TRANSFERS THE BLOCK 4 DATA TO THE AREA POINTED BY AREALOC. THE SUB-BLOCK# IS CONTAINED IN THE COMMAND.

INVOKED PROCEDURES:
BDAM, CONVERT_BINARY, DATA_SET_ERROR, GET_FILE, GET_INTO_PROTECT_KEY_ZERO, GET_OUT_OF_PROTECT_KEY_ZERO, LIBRARY_CRASH, MOVE, PARSING

REFERENCED GLOBALS:
BUFFER, DCB1, LT#, #SUBS, ABLK, #TEXT, ,SUBS, AREALOC, PAGE

LIBRARY_DAOL

DESCRIPTIVE NOTES:
WRITES A LOGICAL TRACK GIVEN THE KEY IN THE COMMAND AND TO THE AREA POINTED BY AREALOC.

IF THE KEY EXISTS, REPLACE IT WITH THE NEW IMAGE. OTHERWISE FIX THE DIRECTORY AND WRITE THE NEW IMAGE.

INVOKED PROCEDURES:
BDAM, CREATE_LABEL_BLOCK, DATA_SET_ERROR, GET_FILE, GETTRK, ISAM, LEADING, LIBRARY_CRASH, MAP, MOVE, PARSING, SYSTEM, WRITE_FILE

REFERENCED GLOBALS:
BUFFER, CURRENT, DCB1, DCB2, CONDCODE, FREE_IMAGE, BUFF, AREALOC, PAGE, LT#, ABLK

LIBBARY_NAME

DESCRIPTIVE NOTES:
CHANGES THE DATA SET NAME IN THE DIRECTORY PAGE TO THE NEW NAME, BUT NOT THE INDIVIDUAL TRACKS, LOCK IN PAGE DATA SET TO -1.

INVOKED PROCEDURES:
INPUTER, LOOK_AT_TABLE, LOOK_AT_PAGE, PARSING, OPERATOR, BDAM, MOVE, WRITE_FILE

REFERENCED GLOBALS:
CONDCODE, COUNT, ADSET, PADDING

PRE_LIBRARY_HASP

DESCRIPTIVE NOTES:
PROVIDES A SINGLE SERVER FOR HASP REQUESTS. THE '/*EOF' IS A HASP INTERVAL READER CONVENTION. MORE THAN ONE '/*EOF' CARDS ARE USED TO FLUSH THE QSAM BUFFER.

INVOKED PROCEDURES:
LIBRARY_HASP, MOVE, OPERATOR, OUTPUTER, ZAP

REFERENCED GLOBALS:
PUT_QUEUE, HASP_USER, TERMAD, COUNT, CONDCODE, BUFFER

PRE LIBRARY_PRINT

DESCRIPTIVE NOTES:
USE THE SINGLE SERVER QUEUE TECHNIQUE BY EXAMINING 'PUT_QUEUE'. IF IT IS BUSY, THEN PASS OVER THE REQUEST. NOTE THAT INITIALLY PRINT, PUNCH, AND HASP SHARE THE QUEUE (SINCE INTERNAL READER CAN BE ALLOCATED TO ONLY ONE DD CARD). PRINT JOB BY GENERATING JCL CARDS. LET THE DEFAULTS BE UH DEFAULTS. NOTE THAT MORE THAN ONE '/*EOF' CARDS ARE PASSED TO FLUSH THE QSAM BUFFERS.

INVOKED PROCEDURES:
LIBRARY_PRINT, MOVEBYTES, MOVER, OPERATOR, OUTPUTER, ZAP

REFERENCED GLOBALS:
PUT_QUEUE, PRINT_USER, TERMAD, COUNT, CONDCODE, BUFFER

PRE LIBRARY_PUNCH

DESCRIPTIVE NOTES:
USE THE SINGLE SERVER QUEUE TECHNIQUE BY EXAMINING 'PUT_QUEUE'. IF IT IS BUSY, THEN PASS OVER THE REQUEST. NOTE THAT INITIALLY PRINT, PUNCH, AND HASP SHARE THE QUEUE (SINCE INTERNAL READER CAN BE ALLOCATED TO ONLY ONE DD CARD). PUNCH JOB BY GENERATING JCL CARDS. LET THE DEFAULTS BE UH DEFAULTS. NOTE THAT MORE THAN ONE '/*EOF' CARDS ARE PASSED TO FLUSH THE QSAM BUFFERS.

INVOKED PROCEDURES:
LIBRARY_PUNCH, MOVEBYTES, MOVER, OPERATOR, OUTPUTER, ZAP

REFERENCED GLOBALS:
PUT_QUEUE, PUNCH_USER, TERMAD, COUNT, CONDCODE, BUFFER

ISS LIBRARY

DESCRIPTIVE NOTES:
INITIALIZES THE SYSTEM, DETERMINES THE COMMANDS, HANDLES THE REQUEST QUEUES. (THREAD THROUGH QUEUE. WHEN THE LAST ONE IS REACHED, START AGAIN. IF THERE ARE NO MORE, WAIT FOR SOME WORK). WATCH OUT FOR DO. MANT TERMINALS.

INVOKED PROCEDURES:
ALLOCATE_DSNODE, FREE_DSNODE, INITIALIZE_DSNODE,
LIBRARY_CREATE, LIBRARY_DADL, LIBRARY_DELETE,
LIBRARY_DISPLAY, LIBRARY_HASP, LIBRARY_INSERT,
LIBRARY_LINK, LIBRARY_LOAD, LIBRARY_NAME,
LIBRARY_SCRATCH, LIBRARY_SELECT, LOGGING, MOVE,
OPERATOR, OUTPUTER, POST, PRE_LIBRARY_HASP,
PRE_LIBRARY_INSERT, PRE_LIBRARY_PRINT,
PRE_LIBRARY_PUNCH, STRING, SUBSTR, SYSTEM, WAIT,
ZAP

REFERENCED GLOBALS:
COUNT, SUBIOTYP, STATUS, IOCOUNT, CONTINUING,
CONDCODE, BUFFER, JOB, NAME, LIBTYPE, IOSTAT,
QUEUE1, POINTER

THE FILE MAINTENANCE ROUTINES CREATE THE LIBRARY FILE, EXTRACTS DISK USAGE DATA FOR THE SYSTEM MANAGER, AND REORGANIZES THE MEMBERS WITHIN THE FILE. THESE PROGRAMS MAINTAIN THE DATA BASE AS AN IBM DIRECT ACCESS DATA SET.

## ZERODISK

ZERODISK INITIALIZES A DIRECT-ACCESS DATA SET WITH THE PROPERLY FORMATTED LOGICAL TRACKS. IT IS WRITTEN IN FORTRAN IV AND USES AN ASSEMBLY LANGUAGE PROGRAM, WRITEQ. WRITEQ IS REQUIRED TO INITIALIZE A BDAM DATA SET WITHOUT KEYS.

ZERODISK HAS TWO PARTS, ONE PART READS THE DATA CARDS, FORMATTING THE PROPER TYPE OF LOGICAL TRACK, AND THE OTHER PART WRITES THE FORMATTED TRACKS ONTO DISK. THE INPUT IS DIVIDED INTO 3 PARTS, ONE SECTION FOR THE DIRECTORY TABLE OF CONTENTS, THE NEXT SECTION FOR THE DIRECTORY PAGES AND THE LAST SECTION FOR FREE LOGICAL TRAC .

EACH SECTION HAS THE SAM STRUCTURE, FOUR BLOCKS PER SECTION. FOR EACH BLOCK THERE IS A FORMAT CARD AND THE APPROPRIATE DATA CARDS FOR THAT BLOCK. THE DESCRIPTION OF EACH BLOCK IS GIVEN IN OS IMPLEMENTATION SECTION.

AFTER ALL THE INPUT CARDS HAVE BEEN READ, AND IF THERE ARE NO ERRORS, THEN THE TRACKS ARE WRITTEN IN THE FOLLOWING MANNER: THE FIRST HALF OF THE DATA SET IS WRITTEN, THEN THE DIRECTORY PAGES, AND FINALLY THE LAST HALF OF THE DATA SET. EACH CYLINDER HAS THE APPROPRIATE LOGICAL TRACKS WITH THE DTOC LOGICAL TRACK ON THE FIRST RECORD OF THE NINETEENTH HEAD.

FOR EACH TYPE OF LOGICAL TRACK, A LISTING OF THE FORMAT CARDS AND THE CORRESPONDING DATA CARDS IS PROVIDED. THE DATA IS IN HEXADECIMAL AS THEY WOULD APPEAR ON THE DISK. FINALLY, THE FIRST 100 WORDS OF THE RECORD ARE DUMPED TO VERIFY THAT THE FORMATTING IS PROPER.

## STATDISK

STATDISK EXAMINES EACH LOGICAL TRACK IN THE UHTSS/2 LIBRARY DATA SET AND PRINTS INFORMATION PERTINENT TO THAT TRACK. THIS PROGRAM, WRITTEN IN FORTRAN, IS INTENDED ONLY FOR THE SYSTEM MANAGERS SINCE IT VIOLATES THE SECURITY LOCKS WITHIN THE SYSTEM.

EACH LOGICAL TRACK IS READ AND CLASSIFIED. DEPENDING UPON ITS CLASSIFICATION, THE APPROPRIATE INFORMATION IS PRINTED. THE MOST IMPORTANT INFORMATION IS THE LOGICAL TRACK POINTERS IN THE DIRECTORIES. ONLY THE CURRENT DTOC IS LISTED. FINALLY SOME STATISTICS REGARDING THE DISK UTILIZATION ARE LISTED. NOTE THAT LOCK IS AT THE MOMENT JUST THE LT# OF SOME LOGICAL TRACK (USUALLY ITSELF EXCEPT IN THE CASE OF DTOCS).

WHILE THE POSTING FORMAT IS QUITE OBVIOUS, ONE PART OF THE LISTING MUST BE CLARIFIED: THE FORMAT OF A BLOCK 3 MAP (AND BLOCK 4) IS THE NUMBER OF SUBBLOCKS, THE NUMBER OF WORDS IN THE DATA AREA, THE NUMBER OF WORDS IN THE KEY AREA, AND THE NUMBER OF WORDS IN THE LOCK AREA.

NOTE THAT THE PERCENTAGE OF TRACKS USED MAY BE THE SAME AS THE SUM OF THE PERCENTAGES FOR PAGE, DTOC, USER LOGICAL TRACKS (FOR THE MOMENT THE ONLY REASON IS THAT INITIALLY SOME TRACKS WERE ALLOCATED FOR FUTURE USE. FOR THE 101 CYLINDER FILE, 2 CYLINDERS WERE SAVED FOR PAGE OVERFLOW TRACKS).

ANLZDISK

THE PURPOSE OF ANLZDISK IS TC PROVIDE A SYSTEMS MANAGEMENT TOOL WHICH MAY BE USED TO EXTRACT UTILIZATION INFORMATION CONTAINED WITHIN THE LIBRARY FILE. THE INPUT IS A SET OF DATA CARDS SPECIFYING WHETHER A GIVEN DATA SET SHOULD BE SAVED OR SCRATCHED. THE OUTPUT IS A LIST OF DATA SETS WHICH WILL EXPIRE WITHIN A WEEK FROM THE DATE THE PROGRAM WAS RUN, A LIST OF DATA SETS WHICH HAVE BEEN SCRATCHED, AND TRACK UTILIZATION FACTS FOR EACH DATA SET. THE TRACK UTILIZATION FACTS CONTAIN THE USE COUNT OF EACH LOGICAL TRACK AND THE NUMBER OF BYTES USED IN THAT TRACK.

THE PROGRAM IS SO STRUCTURED AS TO ENABLE THE UHTSS MANAGER TO EXTRACT ANY INFORMATION CONTAINED ON THE LIBRARY FILE. WHILE AT THE PRESENT TIME THE PROGRAM ALLOWS MAINLY FOR AUTOMATIC CHECKING AND SCRATCHING OF DATA SETS, IT CANBE EEASILY MODIFIED TO EXTRACT SPECIFIC INFORMATION THE SYSTEM MANAGER MAY REQUEST.

THE UTILITY PROGRAM WILL PROVIDE A LIST OF DATA SETS WHICH WILL EXPIRE WITHIN A WEEK FROM THE DATE THE PRORAM IS RUN (USING CHECK_DATE). UNLESS AN EXTENSION OF THE EXPIRATION DATE IS REQUESTED BY THE USER, THE DATA SET WILL THEN BE AUTOMATICALLY SCRATCHED THE NEXT TIME THE ANLZDISK PROGRAM IS RUN (A TWO WEEK GRACE PERIOD IS PROVIDEC HOWEVER, BEFORE ACTUALLY SCRATCHING THE DATA SET). AN OVERRIDE FEATURE IS PROVIDED. THIS FEATURE IS A SCRATCH/SAVE LIST WHICH IS A SET OF DATA CARDS HAVING THE DATA SET NAME IN COLUMN ONE, THE JOB/ACCOUNT NUMBER IN COLUMN TEN, AND THE SCRATCH/SAVE CODE IN COLUMN TWENTY (SCRATCH=1 AND SAVE= ' ').

IN ADDITION TO THE ABOVE FUNCTIONS, THE PROGRAM ALSO PRINTS EACH DATA SET NAME, JOB/ACCOUNT NUMBER, PROGRAMMER'S NAME AND THE ASSOCIATED PAGE WITH THE PRIMARY AND SECONDARY LOGICAL TRACK NUMBERS, AS WELL AS THE NUMBER OF TIMES THE SECONDARY LOGICAL TRACK HAS BEEN USED AND HOW MANY BYTES IT CONTAINS.

THE FUNCTION OF ACHTUNG IS TO PROVIDE A MESSAGE DIRECTORY OF THE SYSTEM. TWO PARAMETERS ARE PASSED, THE FIRST BEING A VARIABLE AND THE SECOND AN ERROR CODE. TEN BASIC CODES ARE PROVIDED DIFFERING BY THE POSITION OF THE VARIABLE WITHIN THE MESSAGE AND THE SEVERITY LEVEL (BLANK (1), NOTE (3), WARNING (3), SEVERE ERROR (3)).

THE FUNCTION OF SCRATCH II IS TO ERASE A SPECIFIED DATA SET AND TO PROVIDE A MESSAGE TO THAT EFFECT. THE PARAMETER IS THE DATA SET NAME.

## ACKNOWLEDGEMENTS

THE AUTHOR WISHES TO THANK THE IBM CORPORATION FOR MAKING IT FEASIBLE FOR THE ALOHA PROJECT TO UNDERTAKE THE DEVELOPMENT OF A TIME-SHARING SYSTEM.

MY DEEPEST APPRECIATION GOES TO DR. W. W. PETERSON, DR. N. ABRAMSON, AND DR. F. KUO WHO PROVIDE THE ATMOSHPERE AND ENCOURAGEMENT REQUIRED IN ORDER TO PERSEVERE IN SUCH AN ARDUOUS ENVIRONMENT.

I WOULD LIKE TO THANK ALICIA NAKAMOTO FOR HER FORTITUDE AND EXCELLENT CRYPTOGRAPHIC ABILITY AND WILHEM BORTELS FOR HIS WORK ON ANLZDISK.

FINALLY I WISH TO NOTE THAT THIS PROJECT MAY SUCCEED IN SPITE OF THE COMPUTING CENTER.

Security Classification

DOCUMENT CONTROL DATA - R & D

*Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified*

| | |
|---|---|
| 1. ORIGINATING ACTIVITY (Corporate author)<br>THE ALOHA SYSTEM<br>University of Hawaii<br>Honolulu, Hawaii 96822 | 2a. REPORT SECURITY CLASSIFICATION<br>UNCLASSIFIED<br>2b. GROUP |

3. REPORT TITLE

UHTSS LIBRARY MANAGEMENT YESTERDAY, TODAY, AND TOMORROW

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Scientific Interim

5. AUTHOR(S) (First name, middle initial, last name)

Alan C.H. Kam

| | | |
|---|---|---|
| 6. REPORT DATE<br>September 1970 | 7a. TOTAL NO. OF PAGES<br>83 | 7b. NO. OF REFS<br>0 |
| 8a. CONTRACT OR GRANT NO.<br>F44620-69-C-0030<br>b. PROJECT NO.<br>9749<br>c. 61102F<br>d. 681304 | 9a. ORIGINATOR'S REPORT NUMBER(S)<br>TR B70-5<br>9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)<br>AFOSR 70-2[illegible]TR | |

10. DISTRIBUTION STATEMENT

1. This document has been approved for public release and sale; its distribution is unlimited.

| | |
|---|---|
| 11. SUPPLEMENTARY NOTES<br>TECH, OTHER | 12. SPONSORING MILITARY ACTIVITY<br>Air Force Office of Scientific Research<br>1400 Wilson Boulevard (SRMA)<br>Arlington, Virginia 22209 |

13. ABSTRACT

This report is a collection of internal reports dealing with the library management. The preliminary design details the anticipated system. The structure of data base reveals the implementation scheme on an IBM 2314 disk facility. Various algorithms are presented to document the actual system conditioned by using XPL and OS/MVT/HASP. Finally a detailed description of the XPL program elaborates upon the modular approach.

DD FORM 1 NOV 1473

Security Classification